CBCS Scheme

USN | | | | | | | | | |

16MCA14

First Semester MCA Degree Examination, Dec.2016/Jan.2017

Computer Organization

Time: 3 hrs. Max. Marks: 80

Note: Answer FIVE full questions, choosing one full question from each module.

Module-1

1 a. Convert the following:
i) $(41)_{10} = (?)_2$
ii) $(0.6875)_{10} = (?)_2$
iii) $(10110001101011)_2 = (?)_{16}$
iv) $(B65F)_{16} = (?)_{10}$
v) $(306.D)_{16} = (?)_2$ **(10 Marks)**

b. Subtract the following:
i) Using 10's complement subtract 72532 – 3250.
ii) Using 2's complement subtract 1010100 – 1000011. **(06 Marks)**

OR

2 a. State the following Boolean postulates:
i) Closure ii) Associate law iii) Commutative law
iv) Identity law v) Inverse vi) Distributive law **(06 Marks)**

b. Express the Boolean function $F = A + B'C$ in sum of minterms. **(04 Marks)**

c. Using K-map simplify the Boolean function $F(w, x, y, z) = \sum(0, 1, 2, 4, 6, 8, 9, 12, 13, 14)$. **(06 Marks)**

Module-2

3 a. Implement the following function using NAND gate:
i) $F = x'y'z' + xyz'$ ii) $xy' + x'y$ **(04 Marks)**

b. Giving circuit diagram, truth table construct a half adder. **(06 Marks)**

c. What is multiplexer? With block diagram and logic diagram, explain 4 to 1 line multiplexer. **(06 Marks)**

OR

4 a. Explain RS flip-flop using NOR gates. **(06 Marks)**

b. What is decoder? Construct a 3 to 8 line decoder. **(10 Marks)**

Module-3

5 a. With neat diagram, explain basic functional unit of a computer. **(06 Marks)**

b. Explain big-endian and little-endian assignments. **(06 Marks)**

c. Explain the basic instruction types. **(04 Marks)**

OR

6 a. What are condition codes? Explain various condition code flags. **(06 Marks)**

b. Explain any five addressing modes. **(10 Marks)**

Module-4

7 a. Write a note on assembler directives. (06 Marks)
b. Explain logical shift instructions. (10 Marks)

OR

8 a. With diagram, explain I/O interface for an input device. (06 Marks)
b. Explain various registers used in DMA interface. (10 Marks)

Module-5

9 a. With a neat diagram, explain 16 × 8 memory organization. (10 Marks)
b. Write a note on RAM. (06 Marks)

OR

10 a. Define ROM cell and explain various types of ROM. (08 Marks)
b. Explain with diagram the connection of memory to the process. (08 Marks)

* * * * *

1. a. i. (41) base 10=(101001) base 2

```
2) 41   1
2)20    0
2)10    0
2)5     1
2)2     0
   1
```

ii. (0.6875) base 10=(0.1011) base 2

0.6$\overline{\underline{87}}$5 * 2 = 1.3750 -> 1
0.3~~7~~50 * 2 = 0.7500 -> 0
0.7~~5~~00 * 2 = 1.5000 -> 1
0.5000 * 2 = 1.0000 -> 1

iii. (10110001101011) base 2=(2C6B) base 16

```
0010   1100   1001   1011
   2         C        6          B
```

iv. (B65F) base 16=(46687) base 10

$B * 16^3 + 6 * 16^2 + 5 * 16^1 + F * 16^0$
= 45056 + 1536 + 80 + 5
= 46687
v. (306.D) base 16 = (001100000110.1101) base 2

3 -> 0011, 0 -> 0000, 6 -> 0110, D -> 1101

b. i. 72532 – 3250 using 10's complement

10's complement of 03250 is (100000–03250) = 96750

72532 + 96750 = 169282, discard the MSB and the answer is 69282

ii. 1010100 - 1000011 using 2's complement

2's complement of 1000011 is 0111101

1010100 + 0111101 = 10010001, discard the MSB and the answer is 0010001

2. a. i. Closure A set S is closed with respect to a binary operator if, for every pair of elements of S, the binary operator specifies a rule for obtaining a unique element of S.

Example: The set of natural numbers N= {1, 2, 3, 4, ...} is closed with respect to the binary operator plus (+) by the rules of arithmetic addition, since a, b∈N we obtain a unique c∈N by the operation a+b= c

ii. Associate A binary operator * on a set S is said to be associative whenever (x* y) * z= x* (y* z) for all x, y, z∈S

iii. Commutative A binary operator * on a set S is said to be commutative whenever x* y= y* x for all x, y∈S

iv. Identity A set S is said to have an identity element with respect to a binary operation * on S if there exists an element e ∈S with the property e* x= x* e = x for any x∈S

example:The element 0 is an identity element with respect to operation + on the set of integers I= {..., −3, −2, −1, 0, 1, 2, 3, ...} since x+ 0 = 0 + x for any x∈I

v. Inverse A set S having the identity element e with respect to a binary operator * is said to have an inverse whenever, for every x∈S, there exists an element y∈S such that x* y= e

vi. Distributive If * and · are two binary operators on a set S, * is said to be distributiveover · whenever x* (y·z) = (x* y) ·(x* z)

b. F = A+B’C

A = A(B+B’)
= AB + AB’
= AB(C+C’) + AB’(C+C’)
= ABC + ABC’ + AB’C + AB’C’

B’C = B’C(A+A’)
= AB’C + A’B’C

F = A+B’C
= ABC+ABC’+AB’C+AB’C’+AB’C+A’B’C
= ABC+ABC’+AB’C+AB’C’+A’B’C

F(A, B, C) = Σ(1,4,5,6,7)

c. f(w,x,y,z) = Σ(0,1,2,4,6,8,9,12,13,14)

1	1		1
1			1
1	1		1
1	1		

g1 = wxy’z’ + wxy’z + wx’y’z’ + wx’y’z
= wy’z(x+x’) + wy’z(x+x’)
= wy’(z+z’)
wy’

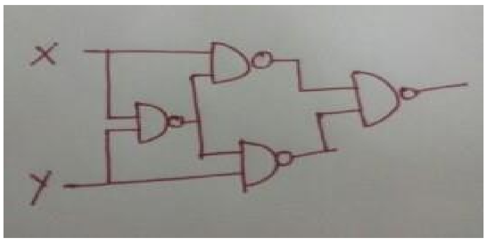
g2 = w’x’y’z’ + w’xy’z’ + w’x’yz’ + w’xyz’
= w’x’z’(y+y’) + w’xz’(y+y’)
=w’z’(x+x’)
=w’z’

g3 = w’xyz’ + wxyz’
= xyz’(w+w’)
xyz’

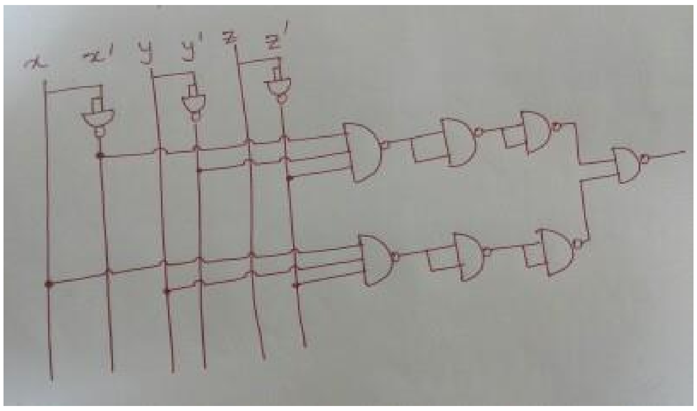
g4 = w’x’y’z’ + w’x’y’z
= w’x’y’

f(w,x,y,z) = wy’+w’z’+xyz’+w’x’y’

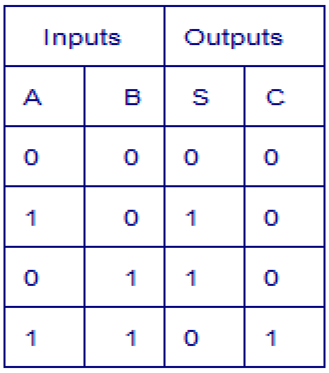
3. a. i.



ii.

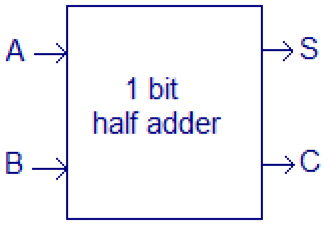


b. Half adder is a combinational arithmetic circuit that adds two numbers and produces a sum bit (S) and carry bit (C) as the output. If A and B are the input bits, then sum bit (S) is the X-OR of A and B and the carry bit (C) will be the AND of A and B. From this it is clear that a half adder circuit can be easily constructed using one X-OR gate and one AND gate. Half adder is the simplest of all adder circuit, but it has a major disadvantage. The half adder can add only two input bits (A and B) and has nothing to do with the carry if there is any in the input. So if the input to a half adder have a carry, then it will be neglected it and adds only the A and B bits. That means the binary addition process is not complete and that's why it is called a half adder. The truth table, schematic representation and XOR//AND realization of a half adder are shown in the figure below.

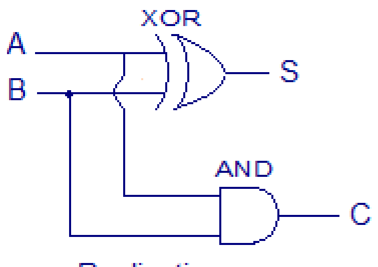


Inputs		Outputs	
A	B	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Truth table



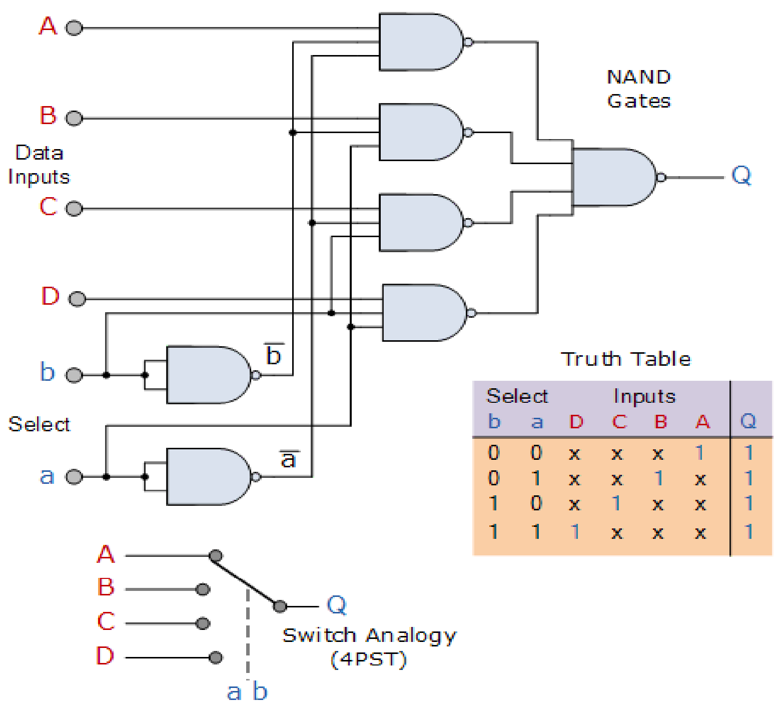
Schematic



Realization

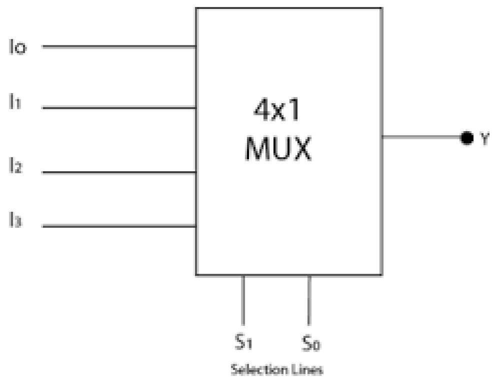
c. The *multiplexer*, shortened to "MUX" or "MPX", is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called "channels" one at a time to the output.

Multiplexers, or MUX's, can be either digital circuits made from high speed logic gates used to switch digital or binary data or they can be analogue types using transistors, MOSFET's or relays to switch one of the voltage or current inputs through to a single output.



Select		Inputs				
b	a	D	C	B	A	Q
0	0	x	x	x	1	1
0	1	x	x	1	x	1
1	0	x	1	x	x	1
1	1	1	x	x	x	1

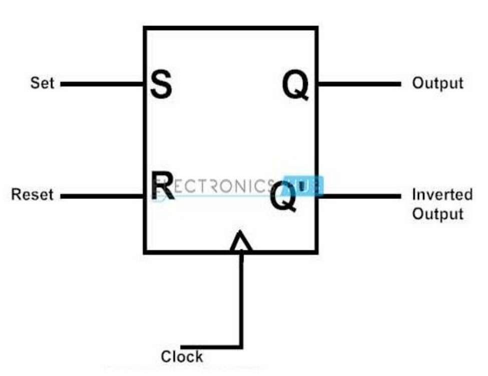
Circuit Diagram and Truth Table of 4x1 MUX



Block Diagram of 4x1 MUX

4. a. The SR flip - flop is one of the fundamental parts of the sequential circuit logic. SR flip - flop is a memory device and a binary data of 1 - bit can be stored in it. SR flip - flop has two stable states in which it can store data in the form of either binary zero or binary one. Like all flip - flops, an SR flip - flop is also an edge sensitive device.

SR flip - flop is one of the most vital components in digital logic and it is also the most basic sequential circuit that is possible. The S and R in SR flip - flop means ‘SET’ and ‘RESET’ respectively. Hence it is also called Set - Reset flip - flop. The symbolic representation of the SR Flip Flop is shown below.

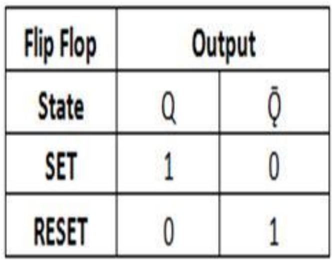


Working

SR flip - flop works during the transition of clock pulse either from low - to - high or from high - to - low (depending on the design) i.e. it can be either positive edge triggered or negative edge triggered.

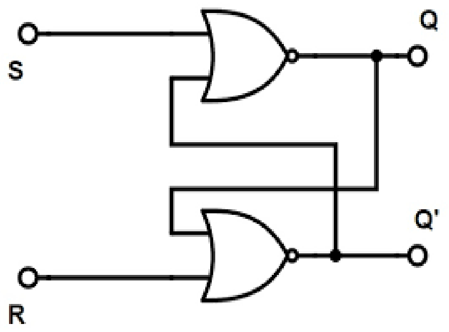
For a positive edge triggered SR flip - flop, suppose, if S input is at high level (logic 1) and R input is at low level (logic 0) during a low - to - high transition on clock pulse, then the SR flip - flop is said to be in SET state and the output of the SR flip - flop is SET to

1. For the same clock situation, if the R input is at high level (logic 1) and S input is at low level (logic 0), then the SR flip - flop is said to be in RESET state and the output of the SR flip - flop is RESET to 0.



Flip Flop	Output	
State	Q	$\bar{Q}$
SET	1	0
RESET	0	1

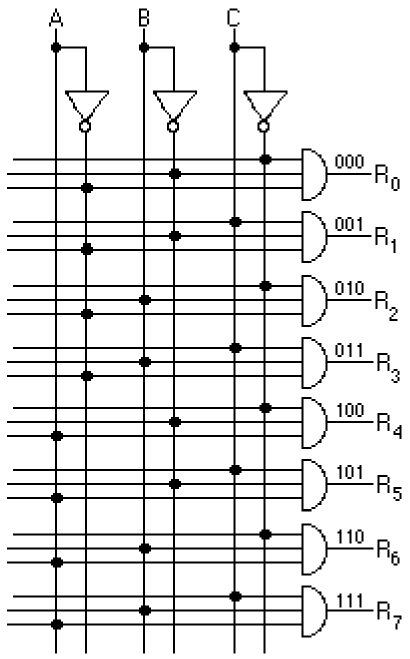
The SR flip - flops can be designed by using logic gates like NOR gates as discussed below



b. A decoder is a circuit that changes a code into a set of signals. It is called a decoder because it does the reverse of encoding, but we will begin our study of encoders and decoders with decoders because they are simpler to design.

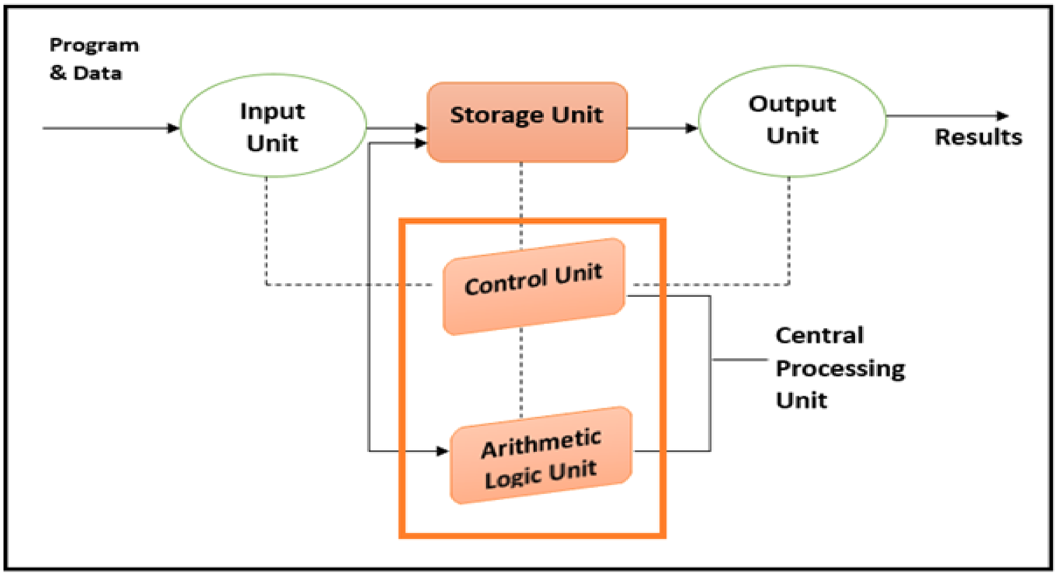
A common type of decoder is the line decoder which takes an n-digit binary number and decodes it into 2^n data lines. The simplest is the 1-to-2 line decoder.

<u>3x8 line decoder</u>



A	B	C	R0	R1	R2	R3	R4	R5	R6	R7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

5. a.



1. Input: This is the process of entering data and programs in to the computer system. You should know that computer is an electronic machine like any other machine which takes as inputs raw data and performs some processing giving out processed data. Therefore, the input unit takes data from us to the computer in an organized manner for processing.

2. Storage: The process of saving data and instructions permanently is known as storage. Data has to be fed into the system before the actual processing starts. It is because the processing speed of Central Processing Unit (CPU) is so fast that the data has to be provided to CPU with the same speed. Therefore the data is first stored in the storage unit for faster access and processing. This storage unit or the primary storage of the computer system is designed to do the above functionality. It provides space for storing data and instructions.

The storage unit performs the following major functions:

- All data and instructions are stored here before and after processing.
- Intermediate results of processing are also stored here.

3. Processing: The task of performing operations like arithmetic and logical operations is called processing. The Central Processing Unit (CPU) takes data and instructions from the storage unit and makes all sorts of calculations based on the instructions given and the type of data provided. It is then sent back to the storage unit.

4. Output: This is the process of producing results from the data for getting useful information. Similarly the output produced by the computer after processing must also be kept somewhere inside the computer before being given to you in human readable form. Again the output is also stored inside the computer for further processing.

5. Control: The manner how instructions are executed and the above operations are performed. Controlling of all operations like input, processing and output are performed by control unit. It takes care of step by step processing of all operations inside the computer.

FUNCTIONAL UNITS

In order to carry out the operations mentioned in the previous section the computer allocates the task between its various functional units. The computer system is divided into three separate units for its operation. They are

1) arithmetic logical unit

2) control unit.

3) central processing unit.

Arithmetic Logical Unit

Logical Unit :After you enter data through the input device it is stored in the primary storage unit. The actual processing of the data and instruction are performed by Arithmetic Logical Unit. The major operations performed by the ALU are addition, subtraction, multiplication, division, logic and comparison. Data is transferred to ALU from storage unit when required. After processing the output is returned back to storage unit for further processing or getting stored.

Control Unit (CU)

The next component of computer is the Control Unit, which acts like the supervisor seeing that things are done in proper fashion. Control Unit is responsible for co ordinating various operations using time signal. The control unit determines the sequence in which computer programs and instructions are executed. Things like processing of programs stored in the main memory, interpretation of the instructions and issuing of signals for other units of the computer to execute them. It also acts as a switch board operator when several users access the computer simultaneously. Thereby it coordinates the activities of computer's peripheral equipment as they perform the input and output.

Central processing unit

The ALU and the CU of a computer system are jointly known as the central processing unit. You may call CPU as the brain of any computer system. It is just like brain that takes all major decisions, makes all sorts of calculations and directs different parts of the computer functions by activating and controlling the operations.

b. <u>Big Endian</u>
In big endian, you store the most significant byte in the smallest address. Here's how it would look:

Address	Value
1000	90
1001	AB
1002	12
1003	CD

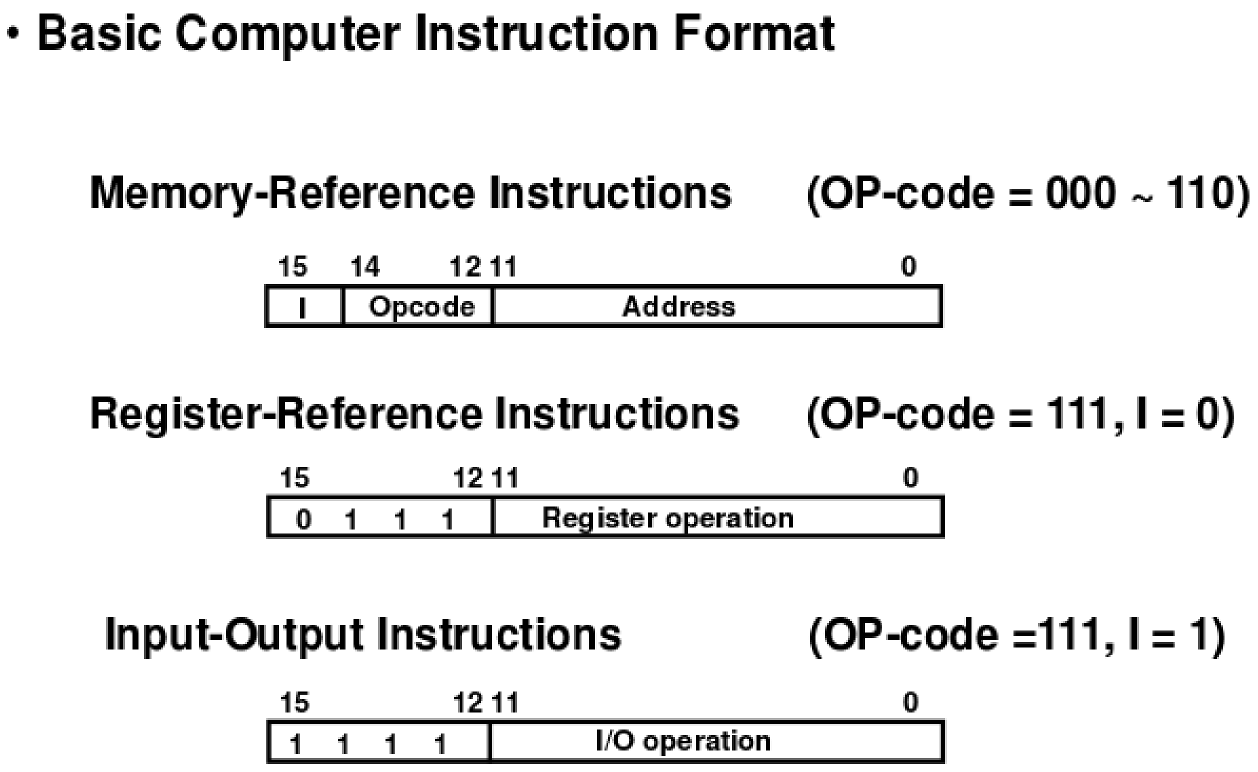
<u>Little Endian</u>
In little endian, you store the *least* significant byte in the smallest address. Here's how it would look:

Address	Value
1000	CD
1001	12
1002	AB
1003	90

Notice that this is in the reverse order compared to big endian. To remember which is which, recall whether the least significant byte is stored first (thus, little endian) or the most significant byte is stored first (thus, big endian).

Notice I used "byte" instead of "bit" in least significant bit. I sometimes abbreciated this as LSB and MSB, with the 'B' capitalized to refer to byte and use the lowercase 'b' to represent bit. I only refer to most and least significant byte when it comes to endianness.

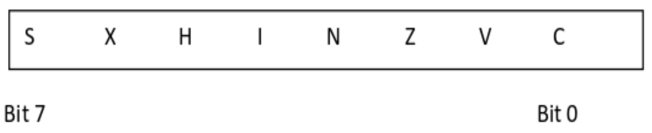
c.
6.
a.



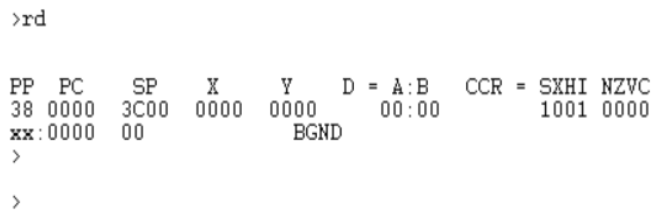
The Condition Code register contains:
· Five flag or status indicators C, H, N, Z, V
· Two interrupt masking bits X, I
· STOP instruction control bit S

The five condition flags are:
· Carry/borrow Flag (C)
· Negativeor Sign Flag(N)
· Zero Flag (Z)
· Overflow Flag (V)
· Half carry Flag (H)

The Flags are either Clear (0) or Set (1)
The bit positions in the CCR are



DeBug 12 Display



```
>rd

PP  PC    SP    X     Y     D = A:B   CCR = SXHI NZVC
38 0000  3C00  0000  0000      00:00         1001 0000
xx:0000  00               BGND
>

>
```

C Status or Carry Flag
The C bit is set when a carry occurs during addition or a borrow occurs
during subtraction.

N Status, Negative or Sign Flag
The N bit shows the state of the MSB of the result. N is most commonly used in two's complement arithmetic, where the MSB of a negative number is 1 and the MSB of a positive number is 0, but it has
other uses. When the MSB of the result of an operation is a 1, then this flag will be set to a 1-otherwise it will be set to a zero

Z Status or Zero Flag
The Z bit is set to a 1 when the result of an operation is zero (all bits are 0) otherwise it will be set to a zero.

V Status or Overflow Flag
The V bit is set when two's complement overflow occurs as a result of an
operation. The V flag is set when the carry out from the most significant bit and the carry in to the most significant bit differ as a result of an arithmetic operation.
V = (Carry In toMSB) ExOR (Carry Out from MSB)

H Status or Half Carry Flag
The H bit indicates a carry from accumulator A (or B) bit 3 during an addition operation. And is often used to correct BCD format. H is updated only by the add accumulator A to accumulator B (ABA), add without carry (ADD), and add with carry (ADC) instructions.

b. The term ***addressing modes*** refers to the way in which the operand of an instruction is specified. Information contained in the instruction code is the value of the operand or the address of the result/operand. Following are the main addressing modes that are used on various platforms and architectures.

1) Immediate Mode

The operand is an immediate value is stored explicitly in the instruction:

Example: SPIM (opcode dest, source)

```
li $11, 3 // loads the immediate value of 3 into register $11

li $9, 8 // loads the immediate value of 8 into register $9

Example : (textbook uses instructions type like, opcode source, dest)

move #200, R0; // move immediate value 200 in register R0
```

2) Index Mode

The address of the operand is obtained by adding to the contents of the general register (called index register) a constant value. The number of the index register and the constant value are included in the instruction code. Index Mode is used to access an array whose elements are in successive memory locations. The content of the instruction code, represents the starting address of the array and the value of the index register, and the index value of the current element. By incrementing or decrementing index register different element of the array can be accessed.

Example: SPIM/SAL - Accessing Arrays

```
.data
array1: .byte 1,2,3,4,5,6
.text
__start:
move $3, $0            # $3 initialize index register with 0
add $3, $3,4           # compute the index value of the fifth element
sb $0, array1($3)      # array1[4]=0
                       # store byte 0 in the fifth element of the array
                       # index addressing mode
done
```

3) Indirect Mode

The effective address of the operand is the contents of a register or main memory location, location whose address appears in the instruction. Indirection is noted by placing the name of the register or the memory address given in the instruction in parentheses. The register or memory location that contains the address of the operand is a pointer. When an execution takes place in such mode, instruction may be told to go to a specific address. Once it's there, instead of finding an operand, it finds an address where the operand is located.

NOTE:

Two memory accesses are required in order to obtain the value of the operand (fetch operand address and fetch operand value).

Example: (textbook) ADD (A), R0

(address A is embedded in the instruction code and (A) is the operand address = pointer variable)

Example: SPIM - simulating pointers and indirect register addressing

The following "C" code:

```
int *alpha=0x00002004, q=5;
*alpha = q;
```

could be translated into the following assembly code:

```
alpha: .word 0x00002004 # alpha is and address variable # address value is
0x00002004
q: .word 5
....
lw $10,q        # load word value from address q in into $10
                # $10 is 5
lw $11,alpha    # $11 gets the value 0x0002004
                # this is similar with a load immediate address value
sw $10,($11)    # store value from register $10 at memory location
                # whose address is given by the contents of register $11
                # (store 5 at address 0x00002004)
```

Example: SPIM/SAL - - array pointers and indirect register addressing

```
.data
array1: .byte 1,2,3,4,5,6
.text
__start:
la $3, array1   # array1 is direct addressing mode
add $3, $3,4    # compute the address of the fifth element
sb $0, ($3)     # array1[4]=0 , byte accessing
                # indirect addressing mode
done
```

4) Absolute (Direct) Mode

The address of the operand is embedded in the instruction code.

Example: (SPIM)

```
beta: .word 2000

lw $11, beta    # load word (32 -bit quantity) at address beta into register $11
                # address of the word is embedded in the instruction code
                # (register $11 will receive value 2000)
```

5) Register Mode

The name (the number) of the CPU register is embedded in the instruction. The register contains the value of the operand. The number of bits used to specify the register depends on the total number of registers from the processor set.

Example (SPIM)

```
add $14,$14,$13     # add contents of register $13 plus contents of
                    # register $14 and save the result in register $14
```

No memory access is required for the operand specified in register mode.

6) Displacement Mode

Similar to index mode, except instead of a index register a base register will be used. Base register contains a pointer to a memory location. An integer (constant) is also referred to as a displacement. The address of the operand is obtained by adding the contents of the base register plus the constant. The difference between index mode and displacement mode is in the number of bits used to represent the constant. When the constant is represented a number of bits to access the memory, then we have index mode. Index mode is more appropriate for array accessing; displacement mode is more appropriate for structure (records) accessing.

Example: SPIM/SAL - Accessing fields in structures

```
.data
student: .word 10000 #field code
.ascii "Smith" #field name
.byte # field test
.byte 80,80,90,100 # fields hw1,hw2,hw3,hw4
.text
__start:
la $3, student  # load address of the structure in $3
                # $3 base register
add $17,$0,90   # value 90 in register $17
                # displacement of field "test" is 9 bytes
                #
sb $17, 9($3)   # store contents of register $17 in field "test"
                # displacement addressing mode
done
```

7) Autoincrement /Autodecrement Mode

A special case of indirect register mode. The register whose number is included in the instruction code, contains the address of the operand. Autoincrement Mode = after operand addressing , the contents of the register is incremented. Decrement Mode = before operand addressing, the contents of the register is decrement.

Example: SPIM/SAL - - simulating autoincrement/autodecrement addressing mode

(**MIPS** has no autoincrement/autodecrement mode)

```
lw $3, array1($17)      #load in reg. $3 word at address array1($17)
addi $17,$17,4          #increment address (32-bit words) after accessing
                           #operand this can be re-written in a "autoincrement like
mode":
lw+ $3,array1($17)         # lw+ is not a real MIPS instruction
subi $17,$17,4             # decrement address before accessing the operand
lw $3,array1($17)
```

7. a. Assembler directives are instructions that direct the assembler to do something
Directives do many things; some tell the assembler to set aside space for variables, others tell the assembler to include additional source files, and others establish the start address for your program. The directives available are shown below:

=

Assigns a value to a symbol (same as EQU)

EQU

Assigns a value to a symbol (same as =)

ORG

Sets the current origin to a new value. This is used to set the program or register address during assembly. For example, ORG 0100h tells the assembler to assemble all subsequent code starting at address 0100h.

DS

Defines an amount of free space. No code is generated. This is sometimes used for allocating variable space.
ID
Sets the PIC’s identification bytes. PIC16C5x chips have two ID bytes, which can be set to a 2-byte value. Newer PICs have four 7-bit ID locations, which can be filled with a 4-character text string.
INCLUDE
Loads another source file during assembly. This allows you to insert an additional source file into your code during assembly. Included source files usually contain common routines or data. By using an INCLUDE directive at the beginning of your program, you can avoid re-typing common information. Included files may not contain other included files. NOTE: The Device Include directive (i.e. INCLUDE 'C:¥PicTools¥16F877.inc') for the targeted device MUST be at the beginning of your source code.
FUSES
NOTE that FUSE CONFIGURATIONs can be '&' together on a single line and/or spread between multiple lines. ALL FUSES directives are ANDed together to create the composite FUSE CONFIGURATION. (view the device "include" file for specific fuse syntax)
IF <expression>
Assembles code if expression evaluates to TRUE.
IFNOT <expression>
Assembles code if expression evaluates to FALSE.
ELSE
Assembles code if preceeding evaluation is rejected.
ENDIF
Ends conditional evaluation.
RESET
Sets the reset start address. This address is where program execution will start following a reset. A jump to the given address is inserted at the last location in memory. After the PIC is reset, it starts executing code at the last location, which holds the jump to the given address. RESET is only available for PIC16C5x chips.
EEORG
Sets the current data EEPROM origin to a new value. This is used to set the data EEPROM address during assembly. This directive usually precedes EEDATA. EEORG is only available for PICs that have EEPROM memory .
EEDATA
Loads data EEPROM with given values. This provides a means of automatically storing values in the data EEPROM when the PIC is programmed. This is handy for storing configuration or start-up information. EEDATA is only available for PICs that have EEPROM memory.

b. **Shift Instructions**

The result of performing an *n* position right logical shift on a binary number containing *m* digits is obtained by:

1. Removing the RIGHTMOST *n* digits from the original number
2. Shifting the remaining digits *n* positions to the right
3. Placing *n* zeros to the left of the resulting number

```
For example, performing a three position, right logical shift on
the number 10110001 results in:


BEFORE:    1 0 1 1 0 0 0 1


AFTER:     0 0 0 1 0 1 1 0



A left shift is performed in a similar manner.


Performing a three position, left logical shift on the number
10110001 results in:


BEFORE:    1 0 1 1 0 0 0 1


AFTER:     1 0 0 0 1 0 0 0
```

Shift Left Logical

```
Format:  label     SLL   R,D(B)
```

The content of the register specified by R are shifted to the left depending on the rightmost six bits of the calculated D(B) address.

The condition code is not altered.

The digits that are shifted off are lost.

```
Suppose that register 7 has the value 0F 0F 0F 0F.  Execution of:

   SLL  R7,2


            0    F    0    F    0    F    0    F

BEFORE:    0000 1111 0000 1111 0000 1111 0000 1111

AFTER:     0011 1100 0011 1100 0011 1100 0011 1100

            3    C    3    C    3    C    3    C
```

```
will change the contents of register 7 to 3C 3C 3C 3C.


Suppose that register 7 has the value 0F 0F 0F 0F and that
register 2 has the value 00 00 00 05.  Execution of:

   SLL  R7,0(R2)


0(R2)  =  0 + 000005  =  000005

The rightmost 6 bits of 0(R2) are 000101.  When converted to
decimal this is 5.


           0    F    0    F    0    F    0    F

BEFORE:   0000 1111 0000 1111 0000 1111 0000 1111

AFTER:    1110 0001 1110 0001 1110 0001 1110 0000

           E    1    E    1    E    1    E    0


will change the contents of register 7 to E1 E1 E1 E0.
```

Shift Right Logical

```
Format:  label     SRL   R,D(B)
```

The content of the register specified by R are shifted to the right depending on the rightmost six bits of the calculated D(B) address.

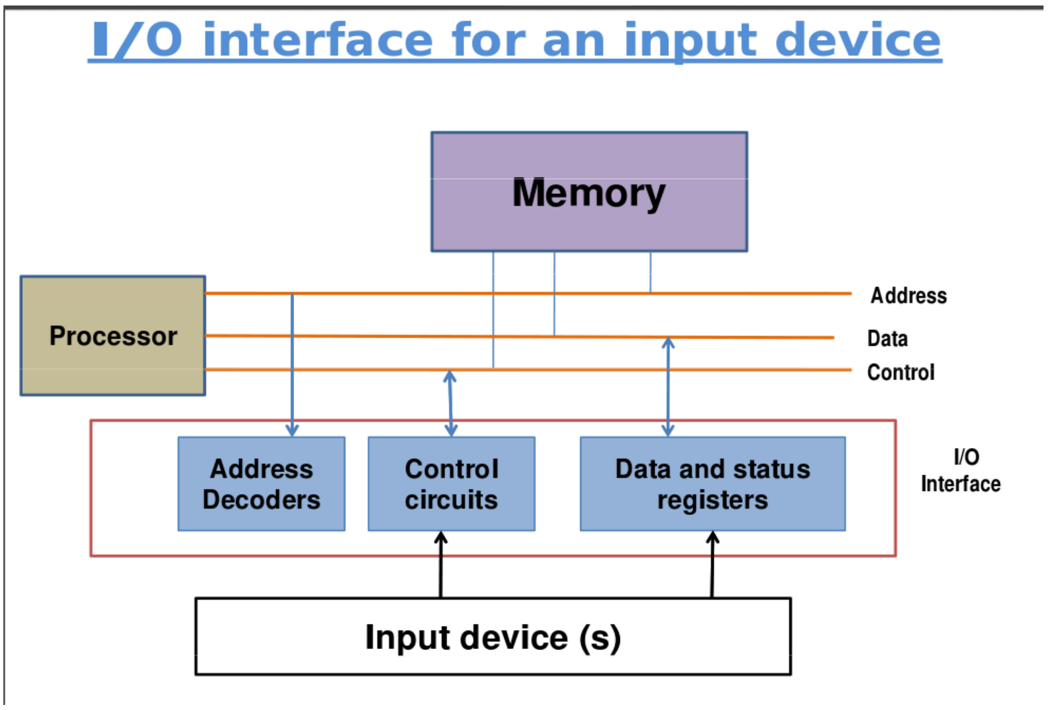
The condition code is not altered.

The digits that are shifted off are lost.

```
Suppose that register 7 has the value 0F 0F 0F 0F.  Execution of:

   SRL  R7,6


           0    F    0    F    0    F    0    F

BEFORE:   0000 1111 0000 1111 0000 1111 0000 1111

AFTER:    0000 0000 0011 1100 0011 1100 0011 1100

           0    0    3    C    3    C    3    C


will change the contents of register 7 to 00 3C 3C 3C.


Suppose that register 7 has the value 0F 0F 0F 0F and that
register 2 has the value 00 00 00 05.  Execution of:

   SRL  R7,0(R2)
```

```
0(R2)  =  0 + 000005  =  000005

The rightmost 6 bits of 0(R2) are 000101.  When converted to
decimal this is 5.


           0    F    0    F    0    F    0    F

BEFORE:   0000 1111 0000 1111 0000 1111 0000 1111

AFTER:    0000 0000 0111 1000 0111 1000 0111 1000

           0    0    7    8    7    8    7    8


will change the contents of register 7 to 00 78 78 78.
```

8. a.



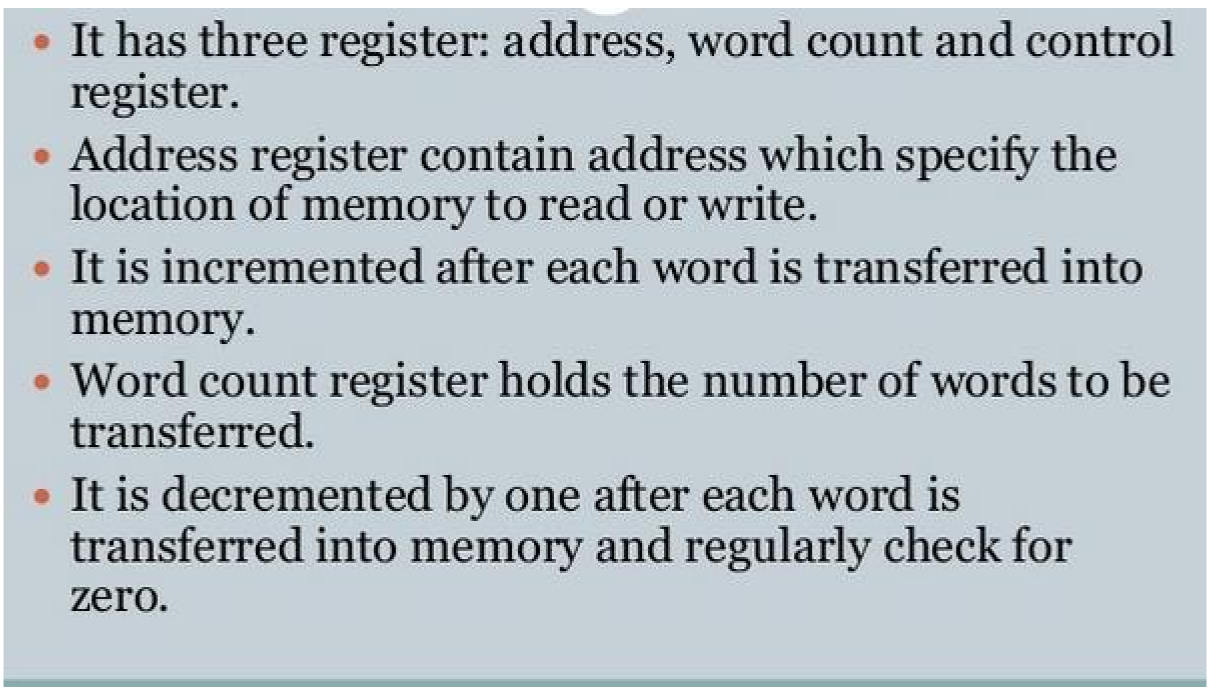
Accessing I/O Devices

· Most modern computers use single bus arrangement for connecting I/O devices to CPU & Memory
· The bus enables all the devices connected to it to exchange information exchange information
· Bus consists of 3 set of lines : Address, Data, Control
· Processor places a particular address (unique for an I/O Dev.) on address lines
· Device which recognizes this address responds to the commands issued on the Control lines commands issued on the Control lines
· Processor requests for either Read / Write
· The data will be placed on Data lines

Hardware to connect I/O devices to bus

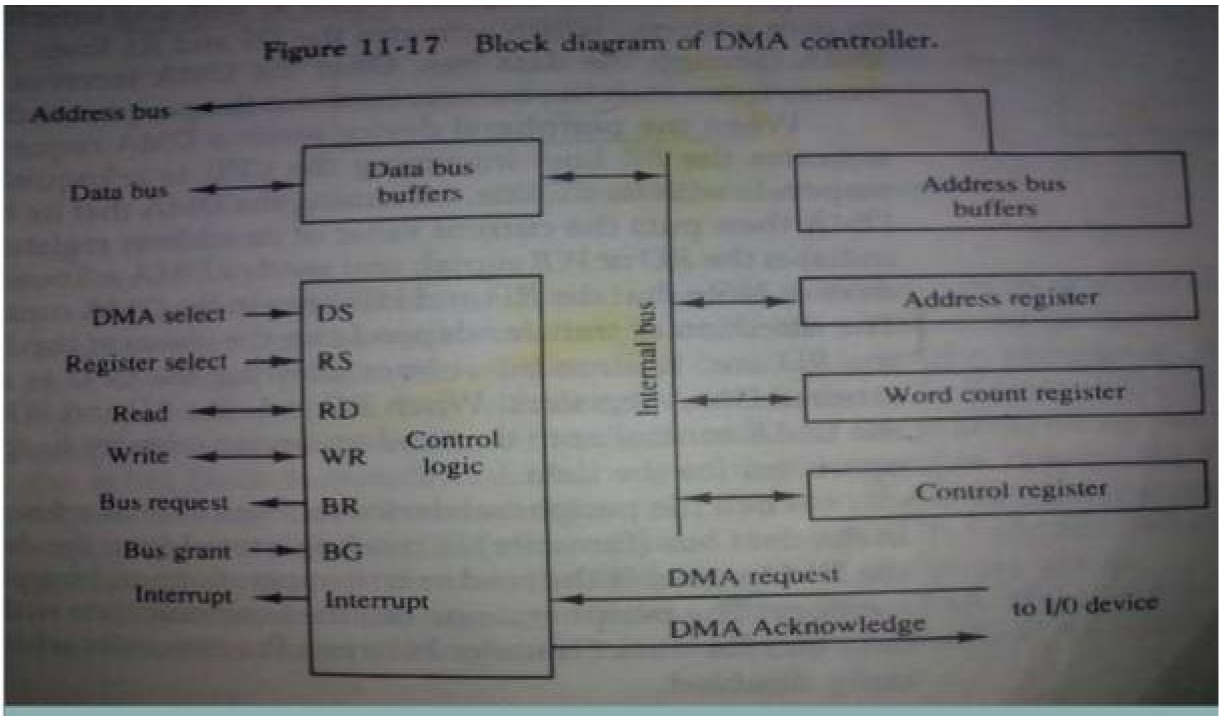
·Interface Circuit
-Address Decoder
-Control Circuits
-Control Circuits
-Data registers
-Status registers
·The Registers in I/O Interface - buffer and control
·Flags in Status Registers like SIN SOUT
·Flags in Status Registers, like SIN, SOUT
·Data Registers, like Data-IN, Data-OUT

b.



- It has three register: address, word count and control register.
- Address register contain address which specify the location of memory to read or write.
- It is incremented after each word is transferred into memory.
- Word count register holds the number of words to be transferred.
- It is decremented by one after each word is transferred into memory and regularly check for zero.

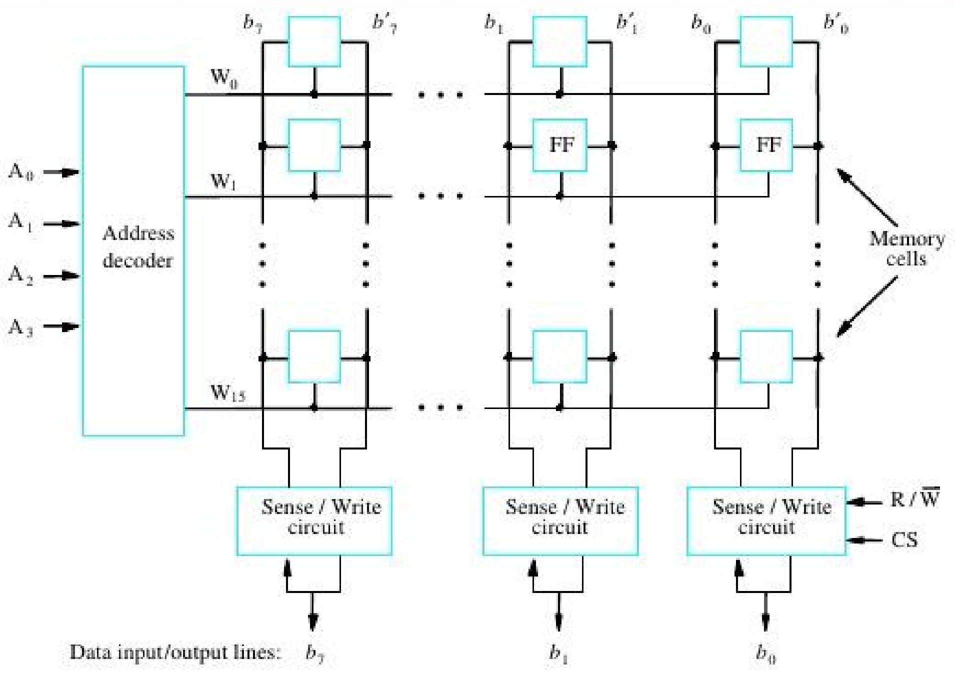
Figure 11-17 Block diagram of DMA controller.



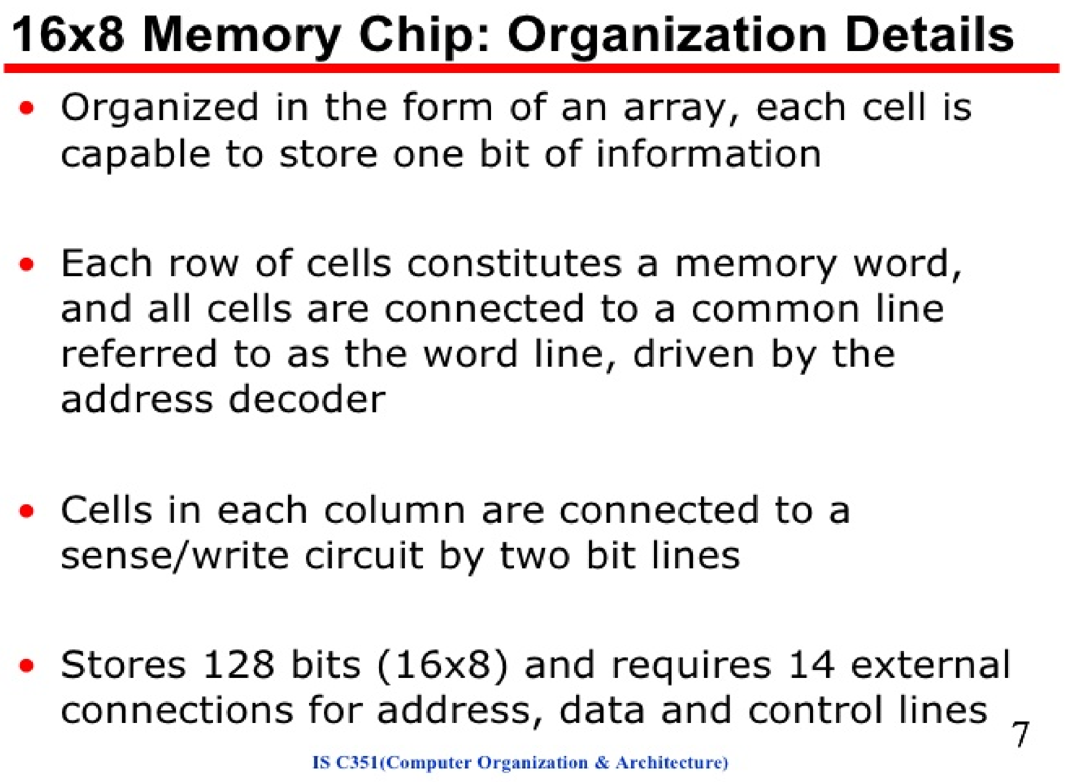
9.

a.

Organization of 16X8 memory chip



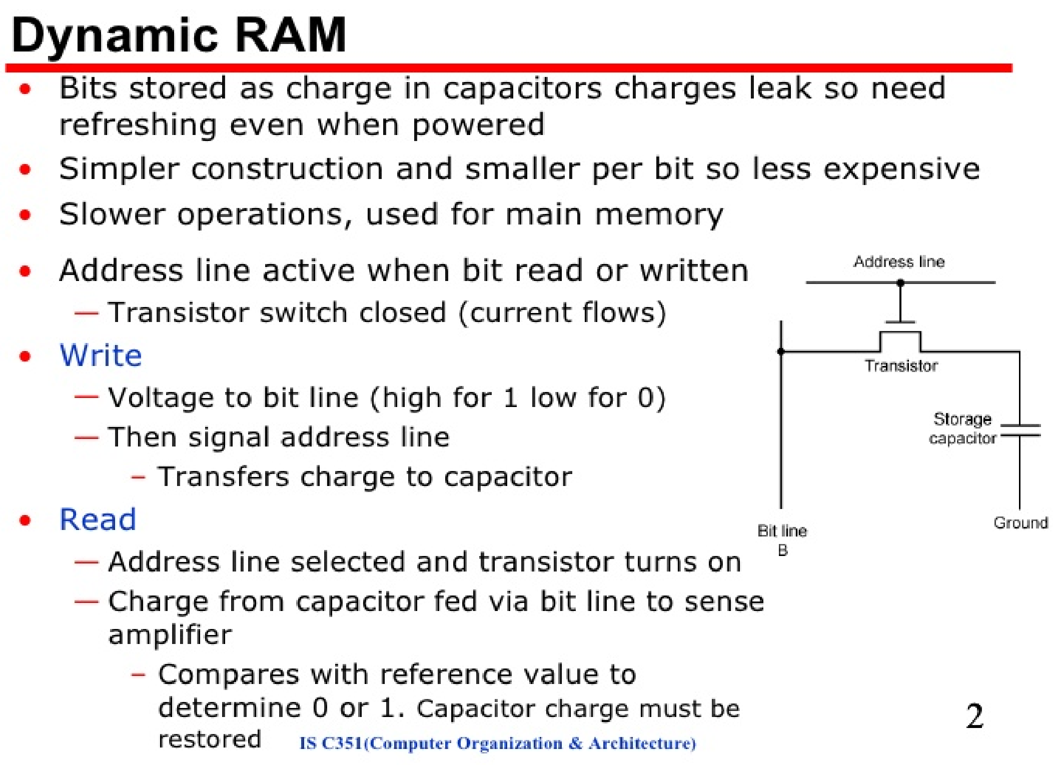
b. **Random-access memory** is a form of computer data storage which stores frequently used program instructions to increase the general



16x8 Memory Chip: Organization Details

- Organized in the form of an array, each cell is capable to store one bit of information
- Each row of cells constitutes a memory word, and all cells are connected to a common line referred to as the word line, driven by the address decoder
- Cells in each column are connected to a sense/write circuit by two bit lines
- Stores 128 bits (16x8) and requires 14 external connections for address, data and control lines

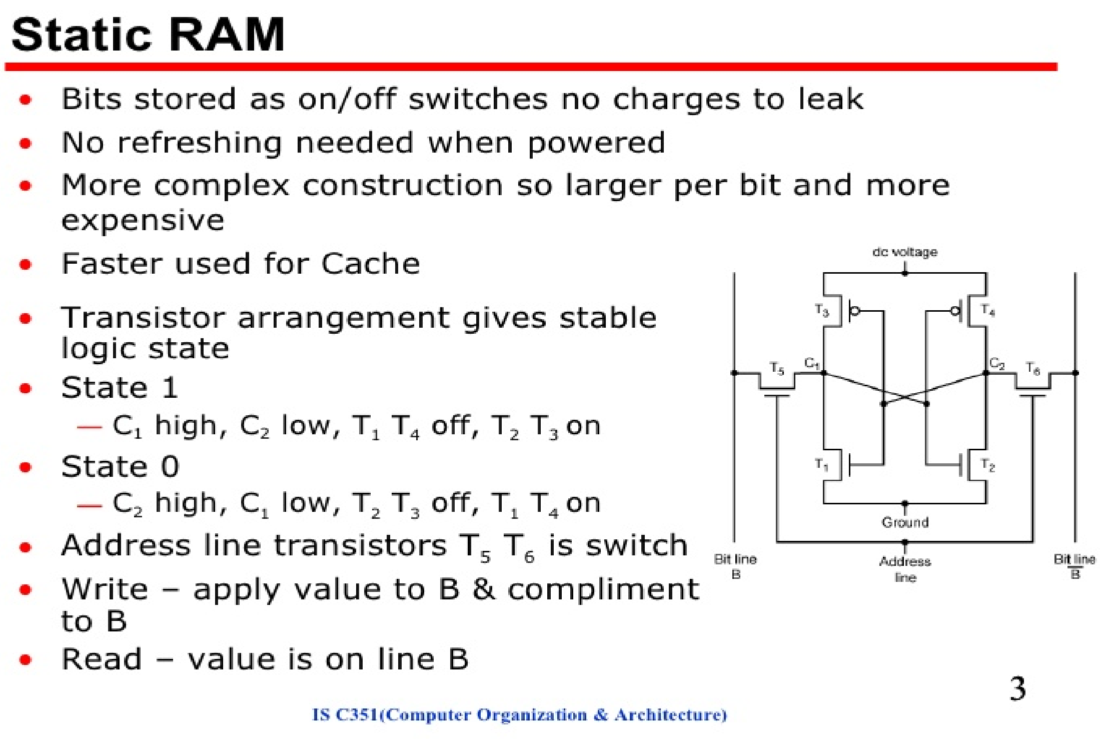
speed of a system. A random-access memory device allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory. In contrast, with other direct-access data storage media such as hard disks, CD-RWs, DVD-RWs and the older drum memory, the time required to read and write data items varies significantly depending on their physical locations on the recording medium, due to mechanical limitations such as media rotation speeds and arm movement.



Dynamic RAM

- Bits stored as charge in capacitors charges leak so need refreshing even when powered
- Simpler construction and smaller per bit so less expensive
- Slower operations, used for main memory
- Address line active when bit read or written
  - — Transistor switch closed (current flows)
- Write
  - — Voltage to bit line (high for 1 low for 0)
  - — Then signal address line
    - – Transfers charge to capacitor
- Read
  - — Address line selected and transistor turns on
  - — Charge from capacitor fed via bit line to sense amplifier
    - – Compares with reference value to determine 0 or 1. Capacitor charge must be restored

10. a. ROM stands for Read Only Memory. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture. A



Static RAM

- Bits stored as on/off switches no charges to leak
- No refreshing needed when powered
- More complex construction so larger per bit and more expensive
- Faster used for Cache
- Transistor arrangement gives stable logic state
- State 1
  - — C_1 high, C_2 low, T_1 T_4 off, T_2 T_3 on
- State 0
  - — C_2 high, C_1 low, T_2 T_3 off, T_1 T_4 on
- Address line transistors T_5 T_6 is switch
- Write – apply value to B & compliment to B
- Read – value is on line B

```
ROM, stores such instructions that are required to start a computer. This operation
is referred to as bootstrap. ROM chips are not only used in the computer but also
in other electronic items like washing machine and microwave oven.
```

Following are the various types of ROM

MROM (Masked ROM)

The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs which are inexpensive.

PROM (Programmable Read only Memory)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM program. Inside the PROM chip there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

EPROM(Erasable and Programmable Read Only Memory)

The EPROM can be erased by exposing it to ultra-violet light for a duration of up to 40 minutes. Usually, an EPROM eraser achieves this function. During programming, an electrical charge is trapped in an insulated gate region. The charge is retained for more than ten years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window(lid). This exposure to ultra-violet light dissipates the charge. During normal use the quartz lid is sealed with a sticker.

EEPROM(Electrically Erasable and Programmable Read Only Memory)

The EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms (milli second). In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of re-programming is flexible but slow.

Advantages of ROM

The advantages of ROM are as follows:

- Non-volatile in nature
- These cannot be accidentally changed
- Cheaper than RAMs
- Easy to test
- More reliable than RAMs
- These are static and do not require refreshing
- Its contents are always known and can be verified