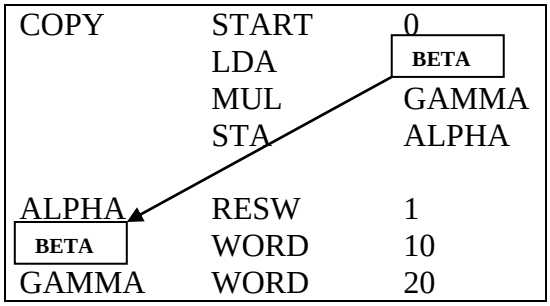
MODULE II

- **Assemblers**
  - **Basic Functions of Assembler**
  - **Assembler output format – Header, Text and End Records**
  - **Assembler data structures**
  - **Two pass assembler algorithm**
  - **Hand assembly of SIC/XE program**
  - **Machine dependent assembler features**
    - **Instruction Format and Addressing Modes**
    - **Program Relocation**

- **Assembler**
  - Assembler is a system software which is used to convert an assembly language program to its equivalent object code.
  - The design of assembler is depends upon the machine architecture.

  - Assembler translates mnemonic operation codes to their machine language equivalents
  - It also assigns machine addresses to symbolic labels

- **<u>Basic Functions of an Assembler</u>**
  1. Convert mnemonic operation codes to their machine language equivalents
     - Eg: Translate STL to 14
  2. Convert symbolic operands to their equivalent machine addresses
     - Eg: Translate the operand RETADR to 1033(address of RETADR)
  3. Convert the data constants to internal machine representations
     - Eg: Translate EOF to 454F46
  4. Build the machine instructions in the proper format
  5. Write the object program and the assembly listing
  - The assembler must also process statements called **assembler directives or pseudo instructions** which are not translated into machine instructions. Instead they provide instructions to the assembler itself
    - **RESB and RESW**- instruct the assembler to reserve memory locations without generating data values.
    - **BYTE** and **WORD –** direct assembler to generate constants as part of the object program
  - All of these functions except number 2 can easily be accomplished by sequential processing of the source program, one line at a time.
  - The difficulty with number 2 is:
    - **Forward reference**: Reference to a label that is defined later in the program.



- If we attempt to translate the program line by line, we will be unable to process this statement because we do not know the address that will be assign to BETA.

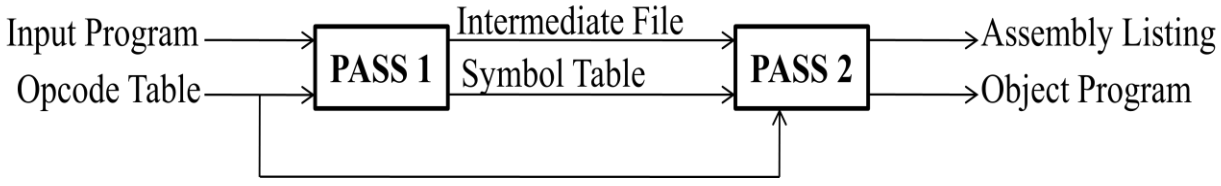
  - Solution: **Introduce 2 passes**
    - First pass: Scan the source program for label definitions and assign addresses
    - Second pass: Perform actual translation

- **<u>Assembler output format</u>(Format of object program)**
  - Assembler must write the generated object code onto some output devices. This object program will later be loaded into memory for execution.
  - The object program format contains 3 records.

	Column	Contents
Header Record	1	H
	2-7	Program name
	8-13	Starting address of object program (HEX)
	14-19	Length of object program in bytes (HEX)
Text Record	1	T
	2-7	Starting address for object code in this record (HEX)
	8-9	Length of object code in this record in bytes (HEX)
	10-69	Object code (HEX, 2 columns per byte of object code)
End Record	1	E
	2-7	Address of first executable instruction in object program (HEX)

- **Assembler Design**
  - An assembler can be
    - Single pass assembler
    - 2 pass assembler or
    - Multipass assembler

- **The functions of the two passes assembler**
  - Pass 1 (define symbols)
    1. Assign addresses to all statements in the program
    2. Save the addresses assigned to all labels for use in Pass 2
    3. Perform some processing of assembler directives
    4. Write intermediate file.
  - Pass 2 (assemble instructions and generate object program)
    1. Assemble instructions (generate opcode and look up addresses)
    2. Generate data values defined by BYTE, WORD
    3. Perform processing of assembler directives not done during Pass 1
    4. Write the object program and the assembly listing



- **Assembler data structures**
  - Assembler uses three main data structures
    - Location Counter(LOCCTR)
    - Operation Code Table(OPTAB)
    - Symbol Table(SYMTAB)
  - **LOCCTR**
    - It is a variable that is used to help in the assignment of addresses.
    - LOCCTR is initialized to be the beginning address specified in the "START" statement
    - After each statement is processed, the length of the assembled instruction or data area to be generated is added to LOCCTR
      - LOCCTR = LOCCTR + (instruction length/size of data area)
    - The current value of LOCCTR gives the address to the label encountered
  - **OPTAB**
    - It must contain the mnemonic operation code and its machine language equivalent.
    - Used to lookup mnemonic operation codes and translate them to their machine language equivalent.
    - It may contain instruction format and length.
    - In Pass 1:
      - OPTAB is used to look up and validate operation code in the source program.
      - Must search the OPTAB to find the instruction length for incrementing LOCCTR.
    - In Pass 2
      - OPTAB is used to translate the operation codes to machine language.
      - It is used to find which instruction format is used.
    - The information in OPTAB is predefined when the assembler itself is written.
    - Implementation
      - Design a special hash table with mnemonic operation code as the key. It provides fast retrieval with minimal searching.
      - It is a static table. Entries are not normally added to or retrieved from it.
  - **SYMTAB**
    - SYMTAB contains name and address for each label in the source program, together with flags to indicate error conditions (Ex: symbols defined in two different places).
    - It may also contain label type, length etc.
    - Pass 1: Labels are entered in to SYMTAB along with their assigned addresses (from LOCCTR)
    - Pass 2: Operands are looked up in SYMTAB to obtain the addresses to be inserted in the assembled instructions.
    - It is a dynamic table
      - Usually organize as a hash table for efficiency of insertion and retrieval.
      - Choose the hash function carefully

- **Two Pass Assembler Algorithm**

```
Algorithm Pass1
{
        Read the input line
        If OPCODE='START'
        {
                starting address = #OPERAND
                LOCCTR = starting address
                Write line to intermediate file
                Read next input line
        }
        Else
                LOCCTR = 0
        While OPCODE != 'END' do
        {
                 Write line to intermediate file along with LOCCTR
                If this is not a comment line
                {
                        If there is a symbol in the label field
                        {
                                Search SYMTAB for LABEL
                                If found
                                        Set error flag(Duplicate Symbol)
                                Else
                                        Insert (LABEL, LOCCTR) into SYMTAB
                        }
                        Search OPTAB for OPCODE
                        If found
                                LOCCTR = LOCCTR + 3
                        Else if OPCODE = 'WORD'
                                LOCCTR = LOCCTR + 3
                        Else if OPCODE = 'RESW'
                                LOCCTR = LOCCTR + 3 x #[OPERAND]
                        Else if OPCODE = 'RESB'
                                LOCCTR = LOCCTR + #[OPERAND]
                        Else if OPCODE = 'BYTE'
                                LOCCTR = LOCCTR + length of constant in bytes
                        Else
                                Set error flags
                }
                Read next input line
        }
        Write last line to intermediate file
        Save (LOCCTR – starting address) as program length.
}
```

Algorithm Pass2

```
{
    Read the first input line from intermediate file
    If OPCODE = 'START'
    {
        Write the line into assembly listing
        Read next input line
    }
    Write Header records to object program
    Initialize first Text record
    While OPCODE != 'END' do
    {
        If this is not a comment line
        {
            Search OPTAB for OPCODE
            If found
            {
                If there is a symbol in OPERAND field
                {
                    Search SYMTAB for operand
                    If found
                        Store symbol value as operand address
                    Else
                        Set error flag
                }
                Else
                    Set 0 as operand address
                Assemble the object code instruction
            }
            Else if OPCODE = 'BYTE' or 'WORD'
                Convert constant to object code
            If object code will not fit into the current Text record
            {
                Write Text record to object program
                Initialize new Text record
            }
            Add object code to Text record
        }
        Write the line into assembly listing along with object code
        Read next input line
    }
    Write last Text record to object program
    Write End record to object program
    Write last listing line
}
```

Example:

Pass 1 Inputs:

Input Program		
PGM1	START	1000
	LDA	ALPHA
	MUL	BETA
	STA	GAMMA
ALPHA	WORD	2
BETA	WORD	4
GAMMA	RESW	1
	END	1000

Opcode Table(OPTAB)	
Opcode	**Hexacode**
LDA	00
MUL	20
STA	0C

Pass 1 Outputs:

Intermediate File			
	PGM1	START	1000
1000		LDA	ALPHA
1003		MUL	BETA
1006		STA	GAMMA
1009	ALPHA	WORD	2
100C	BETA	WORD	4
100F	GAMMA	RESW	1
		END	1000

Symbol Table(SymTAB)	
Label Name	**Label Address**
ALPHA	1009
BETA	100C
GAMMA	100F

Pass 2 Outputs:

Assembly Listing				
	PGM1	START	1000	
1000		LDA	ALPHA	001009
1003		MUL	BETA	20100C
1006		STA	GAMMA	0C100F
1009	ALPHA	WORD	2	000002
100C	BETA	WORD	4	000004
100F	GAMMA	RESW	1	
		END	1000	

Object Program

HPGM1 001000000012
T001000090010092010000C0C100F
T001009060000020000004
E001000

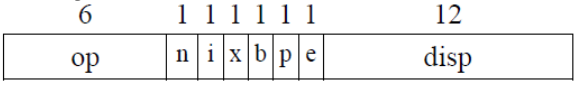
- **Machine Dependent Assembler Features**
  - **Instruction Format and Addressing Modes**
    - **SIC**
      - Word size = 3 bytes
      - Memory size = 2^{15} bytes
      - It has only 5 registers: A,X,L,PC and SW
      - Supports only one instruction format: 3 byte instruction
      - Addressing modes supported: Direct and Indexed
    - **SIC/XE**
      - Memory size = 2^{20} bytes
      - This supports four different types of instruction types
        - 1 byte instruction
        - 2 byte instruction
        - 3 byte instruction
        - 4 byte instruction

- Instructions can be:
  - Instructions involving register to register
  - Instructions with one operand in memory, the other in Accumulator
  - Extended instruction format
- Addressing Modes are:
  - Index Addressing: Opcode m, x
  - Indirect Addressing: Opcode @m
  - PC-relative: Opcode m
  - Base relative: Opcode m
  - Immediate addressing: Opcode #c
  - Extended instruction: +opcode m
- Register-to-register instructions are used wherever possible. Register-to-register instructions are faster than the corresponding register-to-memory operations because they are shorter and do not require another memory reference.
  - Assembler converts the mnemonic operation code to machine language using OPTAB.
  - Assembler changes each register mnemonic to its numeric equivalent. To do this SYMTAB would be preloaded with the register names(A,X, etc.) and their values(0,1, etc.).
    - Eg: COMPR A,S
    - Its object code is A004
- Consider the following statement

```
5       COPY        START       0
```

  - START specifies the beginning of the program
  - 0 indicates it is a relocatable program
- Register to memory instructions
  - Instructions that refer to memory are normally assembled using either program counter relative or base relative mode.
  - The correct target address is calculated by adding the displacement to the contents of Program Counter(PC) or Base Register(B).
    - Program Counter Relative: $-2048 \leq \text{displacement} \leq +2047$
    - Base Relative: $0 \leq \text{displacement} \leq 4095$
  - If displacements are too large, then the 4-byte extended instruction format must be used. It contains 20 bit address field.
  - The assembler directive BASE is used in conjunction with base relative addressing.

- **Program-Counter Relative Addressing Mode**
  - Usually format-3 instruction format is used.



$-2048 \leq \text{disp} \leq 2047$

  - Target Address(TA) = (PC) + disp
  - disp = TA – (PC)

- Eg:

```
0000 FIRST   STL    RETADR
          - - -  -  -  -  -
          - - -  -  -  -  -
0030    RETADR - -  -  -  -- --
```

  - Translate **FIRST STL RETADR**
  - Hex code for STL is 14 => 0001 0100
    - This is 8 bit wide. Delete the rightmost 2 bits and place the remaining part in op fields.
  - TA = address of RETADR (from SYMTAB)= $(0030)_{16}$
  - After fetching this instruction (PC) = $(0003)_{16}$
  - disp = TA – (PC) = $(0030)_{16}$ - $(0003)_{16}$= $(002D)_{16}$
  - Take the rightmost 12 bit disp (02D)
  - If n=i, this instruction is neither indirect nor immediate addressing mode.

Hex	Binary																							
	op						n	i	x	b	p	e	disp/address											
1 7 2 0 2 D	0	0	0	1	0	1	1	1	0	0	1	0	0	0	0	0	0	0	1	0	1	1	0	1

- Eg:

```
0006 CLOOP-  -  -  -
          - - -  -  -  -  -
          - - -  -  -  -  -
0017              J      CLOOP
```

  - Translate **J CLOOP**
  - Hex code for J is 3C => 0011 1100
    - This is 8 bit wide. Delete the rightmost 2 bits and place the remaining part in op fields.
  - TA = address of CLOOP (from SYMTAB)= $(0006)_{16}$
  - After fetching this instruction (PC) = $(001A)_{16}$
  - disp = TA – (PC) = $(0006)_{16}$ - $(001A)_{16}$= $(FFEC)_{16}$
  - Take the rightmost 12 bit disp (FEC)

Hex	Binary																							
	op						n	i	x	b	p	e	disp/address											
3 F 2 F E C	0	0	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	0	1	1	0	0

- **Base Relative Addressing Mode**
  - The base register is under the control of the programmer. The programmer must tell the assembler what the base register will contain during the execution of the program.
  - Base register is used to mention the displacement value.
  - TA = (B) + disp
  - Assembler directives used are:
    - BASE: Informs the assembler that the base register will contain the address of #operand
    - NOBASE: Inform the assembler that the content of base register can no longer be used for addressing.

- Eg:

```
0003            LDB   #LENGTH
                BASE  LENGTH
      ----------------------
      ----------------------
0033  LENGTH    RESW   1
0036  BUFFER    RESW   4096
      ----------------------
      ----------------------
104E            STCH   BUFFER,X
      ----------------------
      ----------------------
```

  - Translate **STCH BUFFER,X**
  - LDB instruction loads the address of LENGTH in the base register.
  - BASE directive explicitly tells the assembler that it has the value of LENGTH.
  - Hex code for STCH is 54 => 0101 0100
    - This is 8 bit wide. Delete the rightmost 2 bits and place the remaining part in op fields.
  - TA = address of BUFFER (from SYMTAB)= $(0036)_{16}$
  - (B) = $(0033)_{16}$
  - disp = TA – (B) = $(0036)_{16}$ - $(0033)_{16}$= $(0003)_{16}$
  - Take the rightmost 12 bit disp (003)

Hex	Binary																							
	op						n	i	x	b	p	e	disp/address											
57C003	0	1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1

- **Immediate Addressing Mode**
  - No memory reference is involved
  - Eg:

```
0020      LDA   #3
```

    - Hex code for LDA is 00 => 0000 0000
    - disp = TA = $(0003)_{16}$

```
   op     n  i  x  b  p  e        disp
000000    0  1  0  0  0  0        003     ⇒  010003
```

- **Immediate with PC-relative mode**
  - Eg:

```
0003            LDB   #LENGTH
      ----------------------
      ----------------------
0033  LENGTH    RESW   1
```

    - Translate **LDB #LENGTH**
    - Hex code for LDB is 68 => 0110 1000
    - TA = address of LENGTH = 0033

- After fetching this instruction (PC) = 0006
- disp = TA – (PC) = $(0033)_{16}$– $(0006)_{16}$ = $(002D)_{16}$

```
   op     n i x b p e      disp
011010   0 1 0 0 1 0      02D    ⇒ 69202D
```

- **Indirect with PC-relative mode**
  - Eg:

```
002A              J       @RETADR
      ----------------------
      ----------------------
0030  RETADR  RESW  1
```

- Translate **J @RETADR**
- Hex code for J is 3C => 0011 1100
- TA = address of RETADR = 0030
- After fetching this instruction (PC) = $(002D)_{16}$
- disp = TA – (PC) = $(0030)_{16}$– $(002D)_{16}$ = $(0003)_{16}$

```
   op     n i x b p e      disp
001111   1 0 0 0 1 0      003    ⇒ 3E2003
```

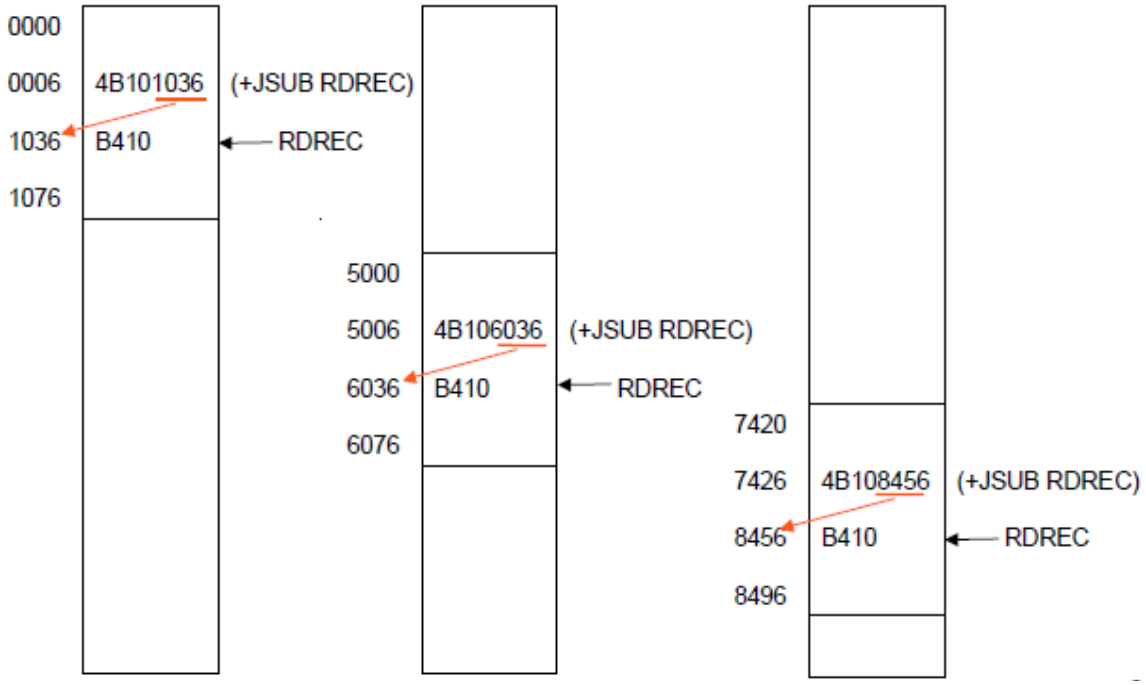
- **Extended Instruction Format**
  - Eg: +LDT #4096
    - Hex code for LDT is 74 => 0111 0100
    - TA = $(01000)_{16}$

```
   op     n i x b p e      disp(20 bits)
011101   0 1 0 0 0 1      01000 ⇒ 75101000
```

- **Program Relocation**
  - Absolute Program
    - The program must be loaded at the address that specified at assembly time.
  - Need for program relocatable
    - Sometimes it is required to load and run several programs at the same time.
    - The system must be able to load these programs wherever there is place in the memory.
    - The exact starting is not known until the load time.

    - Consider the following SIC\XE program

```
      COPY      START   0
0000  ----------------------
      ----------------------
0006  CLOOP   +JSUB     RDREC   4B101036
      ----------------------
      ----------------------
1036  RDREC   CLEAR    X        B410
```



- The above diagram shows the concept of relocation. Initially the program is loaded at location 0000. The instruction JSUB is loaded at location 0006. The address field of this instruction contains 01036, which is the address of the instruction labeled RDREC.
- The second figure shows that if the program is to be loaded at new location 5000. The address of the instruction JSUB gets modified to new location 6036.
- The third figure shows that if the program is relocated at location 7420, the JSUB instruction would need to be changed to 4B108456 that correspond to the new address of RDREC.

- Some instructions do not require modification.
  - Instruction operand is not a memory address.
    - Eg: LDA #3
  - Instruction address is specified using PC relative or Base relative
    - Eg: STL RETADR

      It is assembled using PC or Base relative addressing.
- The only part of the program that require modification at load time are those that specify direct addresses.
- The assembler can identify those part of object program that need modification.
- **Relocatable program** is a program that can be loaded into memory where there is a room, rather than specifying a fixed address at assembly time.
- Relocatable programs use a **Modification record** to store the starting location and the length of the address field to be modified.

- **Modification Record format:**

Mod. Record	1	M
	2-7	Starting location of the address field to be modified, relative to the beginning of the program (HEX)
	8-9	Length of the address field to be modified, in **half-bytes** (HEX)

  - It is placed in between last text record and end record.
  - One modification record is created for each address to be modified.
  - The length is stored in half-bytes (4 bits).
  - The starting location is the location of the byte containing the leftmost bits of the address field to be modified. If the length field contains an odd number of half-bytes, the starting location begins in the middle of the first byte.

- **Object Program Format** is:

	Column	Contents
Header Record	1	H
	2-7	Program name
	8-13	Starting address of object program (HEX)
	14-19	Length of object program in bytes (HEX)
Text Record	1	T
	2-7	Starting address for object code in this record (HEX)
	8-9	Length of object code in this record in bytes (HEX)
	10-69	Object code (HEX, 2 columns per byte of object code)
Mod. Record	1	M
	2-7	Starting location of the address field to be modified, relative to the beginning of the program (HEX)
	8-9	Length of the address field to be modified, in **half-bytes** (HEX)
End Record	1	E
	2-7	Address of first executable instruction (HEX)

- The object program is:

```
HCOPY  000000001077
T000000 1D 17202D 69202D 4B1 01036 032026 290000 332007 4B1 0105D 3F2FEC 032010
T00001D 13 0F2016 010003 0F200D 4B1 0105D 3E2003 454F46
T001036 1D B410 B400 B440 75101000 E32019 332FFA DB2013 A004 332008 57C003 B850
T001053 1D 3B2FEA 1340004F0000F1 B410 774000 E32011 332FFA 53C003 DF2008 B850
T001070 07 3B2FEF 4F000005
M00000705
M00001405
M00002705
E000000
```

Previous Year University Questions

1. List out the basic functions of Assemblers with proper examples.
2. What is meant by forward reference? How it is resolved by two pass assembler?
3. Describe the format of object program generated by the two-pass SIC assembler algorithm
4. Explain the format of the object program generated by a two-pass SIC Assembler, highlighting the contents of each record type.
5. Explain the syntax of the records in the Object Program File.
6. Describe the data structures used in the two pass SIC assembler algorithm
7. Explain the data structures used and their purposes in a two-pass assembler
8. What are the uses of OPTAB and SYMTAB during the assembling process? Specify the uses of each during pass 1 and pass2 of a two pass assembler
9. Give the algorithm for pass 1 of a two pass SIC assembler.
10. With the aid of an algorithm explain the Second pass of a Two Pass Assembler
11. Explain the two passes of the assembler algorithm with proper example.
12. Consider the statements in SIC program. Consider the program being assembled using a 2 pass assembler.

Line no	Location	Label	Opcode	Operand
10	1000	LENGTH	RESW	4
20	----------	NEW	WORD	3

What will be the address value assigned to the symbol NEW during pass 1?

13. Explain with suitable examples, how the different instruction formats and addressing modes of SIC/XE is handled during assembling.
14. Suppose the address associated with the symbol RETADR is 0030 and the machine equivalent code for STL is 14. Assemble the given SIC/XE instruction, by clearly indicating the instruction format, addressing mode and the setting of different flag bits, given the address value assigned to RETADR is 0030.

Location	Label	Opcode	Operand
0000	FIRST	STL	RETADR

15. With suitable example, explain the concept of Program Relocation.
16. Write down the format of Modification record. Describe each field with the help of an example
17. What will happen if a SIC program is loaded in a location different from the starting address specified in the program? Will the program work properly? Justify your answer
18. Explain program relocation with examples. Is there a need to use modification records for the given SIC/XE program segment? Explain your answer. If yes, show the contents of modification record
19. What is a relocatable program? Do all instructions of SIC/XE machine program need modification because of relocation? Justify your answer

```
0000  COPY      START   0
.....................................................
0006              +JSUB   RDREC
000A               LDA    LENGTH
.....................................................
0033  LENGTH    RESW    1
.....................................................
1036  RDREC     CLEAR   X
```