20-1-2022

7. ABSOLUTE LOADER

AIM:

Implement an absolute loader.

THEORY:

- The absolute loader is a kind of loader in which relocated object files are created, loader accepts these files and places them at a specified location in the memory.
- This type of loader is called absolute loader because no relocating information is needed, rather it is obtained from the programmer or assembler.
- The starting address of every module is known to the programmer, this corresponding starting address is stored in the object file then the task of loader becomes very simple that is to simply place the executable form of the machine instructions at the locations mentioned in the object file.
- In this scheme, the programmer or assembler should have knowledge of memory management.

ALGORITHM:

1. Read the first line.
2. If record type = 'H' then store the starting location at loctr.
3. Read next input line.
4. While record type not equal to 'E' do step 5 and 6.
5. Move object code to specified location. Add 3 to loctr.
6. Read next line.

RESULT:

Program was executed and output was verified for different test cases

PROGRAM:

```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
void main()
{
      char rec[20];
      char progname[20];
      char ch;
      int start, length,locctr;
      FILE *objptr,*fp1;
      printf("\n\nThe contents of Input file:\n\n");
     fp1 = fopen("program.txt", "r");
     ch = fgetc(fp1);
     while (ch != EOF)
     {
         printf("%c", ch);
         ch = fgetc(fp1);
     }
     fclose(fp1);

      objptr=fopen("program.txt","r");
      fscanf(objptr,"%s",rec);
      if(strcmp(rec,"H")==0)
      {
           fscanf(objptr,"%s",progname);
```

```
        fscanf(objptr,"%X",&start);
        locctr=start;
        fscanf(objptr,"%X",&length);
        fscanf(objptr,"%s",rec);
    }
    else
    {
        fclose(objptr);
        exit(1);
    }
    printf("\n\n***** Program loaded in memory ****\n");
    printf("\nADDRESS\tOBJECT CODE\n");
    while(strcmp(rec,"E")!=0)
    {
        if(strcmp(rec,"T")==0)
        {
            fscanf(objptr,"%X",&start);
            locctr=start;
            fscanf(objptr,"%X",&length);
        }
        else
        {
            if(locctr>0xFFFF)
                printf("0");
            else if(locctr>0xFFF)
                printf("00");
```

```
                else if(locctr>0xFF)
                    printf("000");
                else if(locctr>0xF)
                    printf("0000");
                else
                    printf("00000");
                printf("%X\t%s\n",locctr,rec);
                locctr+=3;
            }
            fscanf(objptr,"%s",rec);
        }
        fclose(objptr);
}
```

OUTPUT:

```
The contents of Input file:

H COPY 001000 00107A
T 000000 1E 140033 481039 100036 280030 300015 481061 3C0003 20002A 1C0039 30002D
T 002500 15 1D0036 481061 180033 4C1000 801000 601003
E 000000

***** Program loaded in memory ****

ADDRESS OBJECT CODE
000000  140033
000003  481039
000006  100036
000009  280030
00000C  300015
00000F  481061
000012  3C0003
000015  20002A
000018  1C0039
00001B  30002D
002500  1D0036
002503  481061
002506  180033
002509  4C1000
00250C  801000
00250F  601003
```

27-01-2022

8. RELOCATING LOADER

AIM:

Implement a relocating loader.

THEORY:

Loaders that allow for program relocation are called relocating or relative loaders.

Two methods for specifying relocation as part of the object program:

- Modification records: For a small number of relocations required when relative or immediate modes are extensively used. Modification record describe each part of object code that must be changed when program is relocated.
- Relocation bit: Each instruction is associated with one relocation bit indicate the corresponding word should be modified. These relocation bits in Text record is gathered into nit mask. Mostly used for a large number of relocations required when only direct addressing mode can be used in a machine with fixed instruction format.

ALGORITHM:

1. Read the relocation address.
2. While record type not equal to 'E' do step till 7
3. Read input line.
4. If record type = 'H' get variable address, length and go to step 2.
5. Else if record type = T, then get variable address, length and bit mask.
6. Assign relocation bit from bit mask.
7. For i from 0 to length of record type do step 8
8. If relocation bit[i] = 0 then actual address = specified address.
9. Else add the address with relocation address.

RESULT:

Program was executed and output was verified for different test cases

PROGRAM:

```
# include <stdio.h>
# include <string.h>
# include <stdlib.h>
void main()
{
char add[6],length[10],input[10],binary[12],bitmask[12],relocbit,ch;
int start,inp,len,i,address,opcode,addr,actualadd;
FILE *fp1,*fp2;
printf("\nThe contents of Input file:\n\n");
    fp2 = fopen("reloc_input.txt", "r");
    ch = fgetc(fp2);
    while (ch != EOF)
    {
        printf("%c", ch);
        ch = fgetc(fp2);
    }
    fclose(fp2);
printf("\n\nEnter the actual starting address : ");
scanf("%d",&start);
fp1=fopen("reloc_input.txt","r");
printf("\n\n***** Program loaded in memory ****\n");
printf("\nADDRESS\tOBJECT CODE\n");
fscanf(fp1,"%s",input);
while(strcmp(input,"E")!=0)
{
```

```
if(strcmp(input,"H")==0)
 {
 fscanf(fp1,"%s",add);
  fscanf(fp1,"%s",length);
  fscanf(fp1,"%s",input);
  }
   if(strcmp(input,"T")==0)
   {
    fscanf(fp1,"%d",&address);
    fscanf(fp1,"%s",bitmask);
    address+=start;
    len=strlen(bitmask);
    for(i=0;i<len;i++)
    {
    fscanf(fp1,"%d",&opcode);
    addr=opcode%10000;
    opcode=opcode/10000;
    relocbit=bitmask[i];
    if(relocbit=='0')
    actualadd=addr;
    else
    actualadd=addr+start;
    printf("%d\t%d%d\n",address,opcode,actualadd);
    address+=3;
    }
    fscanf(fp1,"%s",input);
```

```
}
}
fclose(fp1);
}
```

OUTPUT:

```
The contents of Input file:

H 1000 200
T 1000 11001 141033 481039 901776 921765 571765
T 2011 11110 231838 431979 891060 661849 991477
E 1000

Enter the actual starting address : 4000


***** Program loaded in memory ****

ADDRESS OBJECT CODE
5000    145033
5003    485039
5006    901776
5009    921765
5012    575765
6011    235838
6014    435979
6017    895060
6020    665849
6023    991477
```