**PROJECT**

**Assembler And CPU**

**Design**

**Submitted by:**

Chirag Kankheria (2019UCO1527)

Kaushal Aggarwal (2019UCO1535)

Ashish Kumar (2019UCO1518)

Chetan Raghav (2019UCO1525)

**DESCRIPTION:**

Assembly language is a low-level programming language which is converted into machine code with the help of an assembler program. We have designed a **One -Pass Assembler** which translates the assembly program into its equivalent binary code and stores it in the memory. The CPU hence can access these instructions from the memory and process them to produce the desired output.

**Language used to code the assembler**: C++

**MEMORY:**

The size of the memory is **256 \* 8** bits. Memory is **byte addressable**.

The memory is divided into 2 segments as follows:

**Code Segment**: Address 0 – 127

**Data Segment**: Address 128-255

**Code Segment:**

This segment stores the code converted by the assembler, instruction is 24 bit long and is stored in memory byte wise.

**Data Segment**:

This segment stores the data(8 bit signed data)

**CPU Design**

**Address bus**: 24 bits

**Data bus**: 8 bits

**Processor registers**(These registers can be accessed by the programmer):

1. **General registers**: R1, R2 ….. R8 are 8 bit registers that can be accessed by the programmers to store the data to perform operations. Values of these registers are stored in data segment. These registers have fixed address and can not be changed by the programmer.
2. **General pointer**: P1, P2, …P8 are 8 bit pointers that can be accessed by the programmers to store the data locations. These have been provided to the user to perform array based problems and memory reference problems like implementing an array or swapping a block of memory. However the pointers store only the address of the data segment only

**Flags**:

**Zero flag:** This flag is set if the data in accumulator is zero, else clear.

**Carry flag**: this flag is set if there is a carry bit

**Sign flag**: this flag is set if the value in AC is negative.

**Overflow flag**: this flag is set if the value in AC is overflowing

**Instructions:**

**Size**: 24 bit

**Operations**: 22 distinct operations

**INSTRUCTION SET**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sno** | **mnemonic** | **opcode** | **RTL** | **Meaning** | **Constraints** | **Example** |
| 1 | MRR | 00000 | R1 <- R2 | move data of R2 in R1 | R1 and R2 belong to general purpose register | MRR R1 R2 |
| 2 | MRM | 00001 | R1 <- M[xxxxxxxx] | move data at 1xxxxxxx location in R1 | xxxxxxx is a 7bit location address(decimal 0-127) and R1 is any general purpose register | MRM R1 64 |
| 3 | MRD | 00010 | R1 <- 25d | move a decimal value in R1 or address in P1 | R1 is any general purpose register and a decimal value from -127 to 127 can be used.  Any memory loacation from 0-127 can be used from data segment | MRD R1 12  MRD P3 64 |
| 4 | MMR | 00011 | M[xxxxxxxx]<-R1 | move data of R1 into memory location 1xxxxxxx | xxxxxxx is a 7bit location address(decimal 0-127) and R1 is any general purpose register | MMR 412  R1 |
| 5 | MPR | 00100 | M[PI]<- R1 | Move data of R1 into memory specified by pointer PI | PI is a pointer that which specifies an 7 bit address | MPR PI R1 |
| 6 | MRP | 00101 | R1<-M[PI] | Move date specified by pointer PI into R1 | PI is a pointer that which specifies an 7 bit address | MRP R1 PI |
| 7 | OAD | 00110 | R1 <- R1 + R2 | add values of R1 and R2 and store them in R1 | R1 and R2 belong to general purpose register | OAD R1 R2 |
| 8 | OSB | 00111 | R1 <- R1-R2 | subtract value of R2 from R1 9and store it in 1-R1 | R1 and R2 belong to general purpose register | OSB R1 R2 |
| 9 | OAN | 01000 | R1 <- R1&R2 | Logical AND values of R1 and R2 and store it in R1 | R1 and R2 belong to general purpose register, value in R1 and R2 should be less than 16 originally | OAN R1 R2 |
| 10 | OOR | 01001 | R1<-R1|R2 | Logical OR of R1 by R2 and store it in R1 | R1 and R2 belong to general purpose register | OOR R1 R2 |
| 11 | OXR | 01010 | R1<-R1 XOR R2 | XOR R1 and R2 and store the result in R1 | R1 and R2 belong to general purpose register | OXR R1 R2 |
| 12 | OMD | 01011 | R1<-R1 % R2 | R1 modulus R2 and store the result in R1 | R1 and R2 belong to general purpose register | OMD R1 R2 |
| 13 | INC | 01100 | R1<-R1+1  P1<- P1+1 | increment R1  OR  Increment P1 | R1 belong to general purpose register | INC R1 |
| 14 | DEC | 01101 | R1<- R1-1 | decrement R1  or  decrement P1 | R1 belong to general purpose register | DEC R1 |
| 15 | CLS | 01110 |  | Circular left shift R1 | R1 belong to general purpose register | CLS R1 |
| 16 | CRS | 01111 |  | Circular right shift R1 | R1 belong to general purpose register | CRS R1 |
| 17 | JIZ | 10000 | PC<-PC+1 | skip the next line if the value of zero flag is reset | next line following this statement should not be HLT | JIZ |
| 18 | CMN | 10001 |  | Complement value of R1 | R1 belong to general purpose register | CMN R1 |
| 19 | XCH | 10010 | R1<-R2 , R2<-R1 | exchange the value of R1 and R2 | R1 and R2 belong to general purpose register | XCH R1 R2 |
| 20 | GTO | 10011 | PC <- xxxxxxxx | jump to a specific line specified by xxxxxxxx | xxxxxxxx should be between the total line number -1. total lines should not exceed 127 | GTO 17 |
| 21 | HLT | 10100 |  | halt the program |  | HLT |
| 22 | BGC |  |  | Marks the beginning address in the memory | Value should be decimal and should be between 0-127 | BGC 15 |

**Internal processor register:**

**Program counter**: 8bits

**Address register**: 8 bits

**Instruction register**: 24 bits

**Data register**: 8 bits

**Assembly program:**

The intstruction can be divided into 3 segments, first is the 8 bit opcode, second is 8 bit register/pointer/memory location and third is 8 bit register/pointer/memory location/data.

For instructions involving 1 address or zero address shall be implemented as shown in the table.

**How does the assembler works?**

The instructions are chosen in such a fashion that assembler works in one pass only, while providing ease and comfortability to user for writing program.

1. The assembler scans every line of the code and separate the literals from each line
2. The first line of the code should be BGC n, where n is any integer randing from 0-125. The location counter and program counter is set to n, if there is any error in the format error is shown to user.
3. First literal in each line is the opcode, assembler checks if the opcode is valid or not, if it is invalid the programs stops running and shows the error to the user along with the line where the error occued.
4. Second literal is the operand1 in major cases. The assembler checks the nature of the opcode and decides what kind of operand is needed for the perfect working of the code, if the operand matches with the desired operand, third operand is checked, else error is returned
5. Then lastly, third literal is scanned and is verified by the assembler based on the opcode. If the operand 2 matches next line is scanned else error is returned.
6. The corresponding binary value is stored in the memory with location counter being incremented after every literal. Incase operand 1 and 2 were not desired then 00000000 is stored in the memory at that location counter.
7. The program should always end with HLT instruction. If HLT is missing then the code will be treated as logically in correct, though no error will be shown.

The values of register are stored into the data segment by the CPU itself and the value is changed the CPU only.

**How to use the assembler?**

First write your code following the rules in input.txt

Run the code

Output will be stored in output.txt file

**What does your assembler returns?**

Our assembler returns the memory of size 256\*8 having the instruction in machine level language that can be fed to the CPU of above mentioned specifications. Assembler also returns the error with line number for the user comfortability.

If there is an error in the code, then the memory will be reset.