

```

int Get_S(char* str, bool& relocatable, Table symbol_table, Table literal_table, ofstream& listing)
{
    if (symbol_table.Is_In_Table(str))
    {
        return (symbol_table.Get_Value(str));
    }
    else if (literal_table.Is_In_Table(str))
    {
        return literal_table.Get_Value(str);
    }
    else
    {
        if ((str[0] != '0') && (str[0] != '1') && (str[0] != '2') && (str[0] != '3') && (str[0] != '4') && (str[0] != '5') && (str[0] != '6') && (str[0] != '7') && (str[0] != '8') && (str[0] != '9') && (str[0] != '-'))
        {
            listing << "\nUnknown symbol \"" << str << "\"\n";
            cerr << "Program exited abnormally\n";
            exit(1);
        }
        relocatable = false;
        return atoi(str);
    }
}

int Get_R(char* str, Table symbol_table, Table literal_table, ofstream& listing)
{
    if (symbol_table.Is_In_Table(str))
    {
        return symbol_table.Get_Value(str);
    }
    else if (literal_table.Is_In_Table(str))
    {
        return literal_table.Get_Value(str);
    }
    else
    {
        if ((str[0] != '0') && (str[0] != '1') && (str[0] != '2') && (str[0] != '3') && (str[0] != '4') && (str[0] != '5') && (str[0] != '6') && (str[0] != '7') && (str[0] != '8') && (str[0] != '9') && (str[0] != '-'))
        {
            listing << "\nUnknown symbol \"" << str << "\"\n";
            cerr << "Program exited abnormally\n";
            exit(1);
        }
        return atoi(str);
    }
}

bool Is_Literal(char* str)
{
    if((strlen(str) < 2) || (str[0] != '='))
        return false;
    else
        return true;
}

bool In_Addr_Range(int i)
{
    if(i < 0 || i > 255)
        return false;
    else
        return true;
}

void Pass_Two(ifstream& source, ifstream& middle, ofstream& obj, ofstream& listing, Table symbol_table, Table literal_table)
{
    Table op_table;
    char SegmentName[6];
    SegmentName[6] = '\0';

```

```

int Initial_Load_Address, First_To_Execute, Segment_Length;
char* m;
bool Is_Relocatable;
middle >> SegmentName;
middle >> m;
if (strcmp(m, "M") == 0)
{
Initial_Load_Address = 0;
Is_Relocatable = true;
}
else
{
Initial_Load_Address = atoi(m);
Is_Relocatable = false;
}
middle >> First_To_Execute;
middle >> Segment_Length;
obj << 'H';
obj.width(2);
obj.fill('0');
obj << hex << (Initial_Load_Address + First_To_Execute);
obj.flags(ios::left);
obj.width(6);
obj.fill(' ');
obj << SegmentName;
obj.flags(ios::right);
obj.width(2);
obj.fill('0');
obj << hex << Initial_Load_Address;
obj.width(2);
obj.fill('0');
obj << hex << Segment_Length;
if (Is_Relocatable)
{
obj << 'M';
}
obj << '\n';
if ((Initial_Load_Address + Segment_Length) > 255)
{
listing << "Invalid memory address attempted";
cerr << "Program exited abnormally\n";
exit(1);
}
int location_counter = Initial_Load_Address;
//literal_table.Update_Values(Initial_Load_Address);
int n = 1;
listing << "\n#      Label    Op    Operands          |Loc Op R X S Reloc\n"
<< "-----\n";

while(!source.eof())
{
Is_Relocatable = (strcmp(m, "M") == 0);
char buffer[80];
int data = 0;
bool output = true;
do
{
source.getline(buffer, 80);
}while(buffer[0] == ';');
if (strlen(buffer) > 0 && buffer[0] != '\n')
{
listing.flags(ofstream::left | ofstream::dec);
listing.width(5);
listing << n;
n++;
char* token;
token = strtok(buffer, " ");
if (!op_table.Is_In_Table(token))

```

```

{
listing.width(8);
listing << token;
token = strtok(NULL, " ");
}
else
{
listing << "          ";
}
if (!op_table.Is_In_Table(token))
{
listing << "\nUnknown instruction \"" << token << "\"\n";
cerr << "Program exited abnormally\n";
exit(1);
}
listing.width(5);
listing << token;
char* token2;
if (strcmp(token, "CCD") == 0)
{
buffer[15] = '.';
token2 = strtok(NULL, ".");
}
else
{
token2 = strtok(NULL, " ");
}
if (token2 != NULL)
{
listing.width(18);
listing << token2;
}
else
{
listing << "          ";
}
}
listing.width(1);
listing << '|';
if (op_table.Get_Value(token) > 15)
{
if (strcmp(token, "NMD") == 0)
{
data = Get_S(token2, Is_Relocatable, symbol_table, literal_table, listing);
if (Is_In_Range(data)==false)
{
listing << "\n\nOperand in NMD pseudo_op not an integer in range";
cerr << "Program exited abnormally\n";
exit(1);
}
location_counter++;
}
else if (strcmp(token, "CCD") == 0)
{
data = token2[1];
data *= 256; //left-shift 8
data += token2[2];
data *= 16; //left-shift 4
location_counter++;
}
else if (strcmp(token, "RES") == 0)
{
int value = atoi(token2);
if(value < 1 || value > 255)
{
listing << "\n\nOperand in RES pseud_op not an integer in range";
cerr << "Program exited abnormally\n";
exit(1);
}
}
}

```

```

    }
    location_counter += value;
    output = false;
}
else
{
    output = false;
}
Is_Relocatable = false; //the "S" field of any pseudo-op
                        //should NOT be relocatable
}
else
{
    listing.width(4);
    listing << hex << (location_counter);
    listing.width(3);
    listing << hex << op_table.Get_Value(token);
    int z = strlen(token2);
    token2[z] = '.';
    token2[z+1] = '\0';

    data = op_table.Get_Value(token); //put the op code into data
    data *= 16; //left-shift 4
    char* temp_tok;
    temp_tok = strtok(token2, ","); //get the R value
    if (temp_tok == NULL)
    {
        listing << "\nIllegal operand\n";
        cerr << "Program exited abnormally\n";
        exit(1);
    }
    //err check - if(Is_Literal(temp_tok)),
    //then this is an error, R field can't be a literal
    if(Is_Literal(temp_tok))
    {
        listing << "\nIllegal operand \"" << temp_tok << "\" in R field";
        cerr << "Program exited abnormally\n"; //is this really fatal?
        exit(1);
    }
    int R = Get_R(temp_tok, symbol_table, literal_table, listing);
    listing.width(2);
    listing << hex << R;
    //int R = atoi(temp_tok);
    //err check - if(R < 0 || R > 3), R out of range
    if(R < 0 || R > 3)
    {
        listing << "\nR field integer \"" << R << "\" not in range";
        cerr << "Program exited abnormally\n"; //is this really fatal?
        exit(1);
    }
    data += R;
    data *= 16; //left-shift 4
    temp_tok = strtok(NULL, ".");
    if (temp_tok == NULL)
    {
        listing << "\nIllegal operand";
        cerr << "Program exited abnormally\n";
        exit(1);
    }
    if (temp_tok[(strlen(temp_tok)-1)] == ')')
    {
        if (temp_tok[0] == '=')
        {
            listing << "\nAttempt to index a literal\n";
            cerr << "Program exited abnormally\n";
            exit(1);
        }
        char* temp_tok2;

```

```

temp_tok2 = strtok(temp_tok, "(");
int S = Get_S(temp_tok2, Is_Relocatable, symbol_table, literal_table, listing)
;

//err check - make sure 0<=S<=255
if(! In_Addr_Range(S))
{
listing << "\nS field integer \"" << dec << S << "\"" not in range";
cerr << "Program exited abnormally\n"; //is this really fatal?
exit(1);
}
temp_tok2 = strtok(NULL, "(");

//err check - if(Is_Literal(temp_tok2)||symbol_table.Is_In_Table(temp_tok2)),
//then this is an error, X field can't be a literal
if(Is_Literal(temp_tok2))
{
listing << "\nIllegal operand \"" << temp_tok2 << "\"" in X field";
cerr << "Program exited abnormally\n"; //is this really fatal?
exit(1);
}
int X = atoi(temp_tok2);
listing.width(2);
listing << hex << X;
//err check - if(X < 0 || X > 3), X out of range
if(X < 0 || X > 3)
{
listing << "\nX field integer \"" << X << "\"" not in range";
cerr << "Program exited abnormally\n"; //is this really fatal?
exit(1);
}
data += (X*4);
data *= 256; //left-shift 8

if((strcmp(token, "BR") == 0)&&(R == 0))
{
//this isn't a relocatable OP
Is_Relocatable = false;
}
else
{
//update S(X) on all OP but BR R=0.
S += Initial_Load_Address;
}
data += S;
listing.width(3);
listing << hex << S;
}
else
{
data *=256; //left-shift 8, b/c there is no X
//field
int X = 0;
listing.width(2);
listing << hex << X;
int S = Get_S(temp_tok, Is_Relocatable, symbol_table, literal_table, listing);
listing.width(3);
if((strcmp(token, "BR") == 0)&&(R == 0))
{
//this isn't a relocatable OP
Is_Relocatable = false;
listing << hex << S;
}
else
{
//update S(X) on all OP but BR R=0.
listing << hex << S+Initial_Load_Address;
}

//err check - make sure 0<=S<=255
if(! In_Addr_Range(S))
{

```

```

        listing << "\nS field integer \"" << dec << S << "\" not in range";
        cerr << "Program exited abnormally\n"; //is this really fatal?
        exit(1);
    }
    data += S;
    data += Initial_Load_Address;
}
location_counter++;
}
if (output)
{
    obj << 'T';
    obj.width(2);
    obj.fill('0');
    obj << hex << (location_counter-1);
    obj.width(5);
    obj.fill('0');
    obj << hex << data;

    if (Is_Relocatable)
    {
        obj << 'M';
        listing << 'M';
    }
    obj << '\n';
}
listing << '\n';
}
}
literal_table.Put_Literals(obj, location_counter);
listing << "\nSymbol Table:\n";
symbol_table.Put_Table(listing);
listing << "\nLiteral Table:\n";
literal_table.Put_Table(listing);
}

```