

**Foundations To Computer Systems Design**  
**Professor V. Kamakoti**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology Madras**  
**Module 5.4**  
**Assembler: Pass 1**

(Refer Slide Time: 0:16)

Module 5.4

Pass 1: Assembler → Assemble myprog.asm  
 myprog.inter → Pass 2  
 myprog.hack

Initialize Symbol table  
 Remove Comments  
 Remove whitespace  
 Find type of command

$D=A+M$   
 D=A+M // Updating Memory (Loop?)

Module 5.4: Assembler: Pass 1

PROF. V. KAMAKOTI  
 IIT Madras

Welcome to module 5.4 in this 5.4 we will talk about the coding of Pass 1 of the assembler. So what is required as we see here is that, so in your system you just enter assembler my prog.asm now what we will do is in Pass 1 this will create an intermediate file called myprog.inter that will be taken again as input by Pass 2 and it will create your myprog.hack which is the 16-bit binary.

So this is the 2 pass assembly stages, so what we need to do as a part of Pass 1 already we have explained that it means to populate the symbol table. It has to look for symbols and for all the symbols that are seeing there, it needs to find a mapping to an address and that address is what the Pass 2 will do to basically get the binary, right?

So what will Pass 1 do, it will read line by line the code first it will initialize symbol table what is initialization of the symbol table we have seen in module 5.3, right? So s there will be some 23 entries we have to do that and the remaining things have to be null and minus 1 all those things we have seen. Now every line what it does, this particular Pass 1 will take your myprog.asm file and it will read every line.

And in that line if there are comments it will remove first it will remove whitespace, suppose I have say D equal to A plus M lot of whitespace still your system has to allow this, right? So

we will now make it as D equal to A plus M removing all the whitespaces that are you know at this bar, this bar, this bar, this bar. So we now remove the whitespace after removing the whitespace you just have to comment.

Now at that point you could have and after you remove the whitespace now you have to see if it is a comment or not, the 2<sup>nd</sup> thing is to remove a comment. So comment there are 2 challenges one thing is I could start with a comment everything is a comment that, in the assembly. Then you ignore that entire line as I told you comments have to be ignored there is no binary equivalent for a comment, okay.

So we just ignore it, the other thing is I could have D equal to A plus M and I could have something like updating memory. So when we do this what will happen is, when you you know remove all the whitespaces this would have become D equal to A plus M//updating memory, right? So you is just there where comment the first 2 characters itself would be //and we could because we have removed all the whitespaces then you can ignore the entire comment.

But in this case I will be just scanned till D equal to A plus M, the first time I see the 2 double / just do not pass anything beyond this and so we are basically reading a string, what you read? You read line by line and put it as a string, so you just make, so what would have happened? let us say this is a pass comment, reading, so what would have happened here is.

This will be pass comment of 0, 1, 2, 3, 4 D equal to a plus M, right? There 5 characters already there then the 6th and 7th you see it is a comment just you make the 5<sup>th</sup> character as the null character. So the string and here D equal to A plus M, alright. So that is something we need to be very careful about comment, okay. So you remove whitespace you remove comment.

For every line by line you read remove whitespace remove comment, now the next thing is now we have to find the type of comment. Now since I've removed all the whitespaces and also the comments i so if I say the first character is going to be this parenthesis then obviously this is going to be a label comment and what you call as an L comment. So that is a label that means for Pass1 it is very very important.

I have to take whatever symbol is there within that label and try to add the entry and do, right? So that's an L comment, now to make this a little bit robust I could have this loop, before we go there, so this is one L comment another thing is that the first character, since

again, so look I have removed all the whitespaces, right? If the first character is at that you have read then you know that this is a A comment.

Then you go and check whether this is a constant, so whatever is following till the end of line in this particular string please check if it is going to be zeroes and ones digits than it essentially becomes a constant. If the first thing start with an alphabet or anything other than zeros and one, 01 anything other than a numeric number then you know that this is again a symbol.

So then you have at symbol now we have to go to that symbol table and see if add it and do whatever we have described earlier, right?

So now we know if the L comment I know if it is an add comment with a symbol I have to take that symbol and keep adding it to the symbol table. Now the last thing is that to make it a little more robust I could have something like this.

(Refer Slide Time: 6:38)

Module 5.4: Assembler: Pass 1

PROF. V. KARAKOTI  
IIT Madras

I could have a symbol say loop and then I could have some at right? At i I could have loop and I will have at 100, I will have loop and I will have some D equal to D plus M or something, so I could have an L followed by an A which has a symbol, A also has a symbol or L followed by A with as a constant or L followed by a C type instruction, right? You know where we have A type and C type instructions, right?

So in this case again since we have removed all the whitespaces etc this will be looking like loop immediately I will have an @ and then some or I will have loop and immediately I will

have a `a(0)(27:33)`, right? So this also to just make your you know assembler little more robust we introduce this so then basically we need to understand this loop this symbol and then if this is a symbol both of these symbols have to be updated in the symbol table.

And in the symbol table you'll find whether address is already entered with address minus 1 already entered with some address already fixed, so these are the things. So this is how we resolve the symbols as a part of Pass 1. So after doing this line by line and after we reached the end of the file, alright so we basically go and check the last thing that we see is that there will be some more symbols for which we have not assigned address and those are the non-labels those are the data, right?

We have already seen in module 5.2 and those data points we basically go and put address starting from something like 16 or 17, we just put those addresses and fill it up, right? So this is what the basic table is. This is what the basic Pass 1 of your assembler is. So at the end of this you will have a uncommented or comment removed D commented that's correct word, a de commented de whitespace file where in all your comments and whitespace removed.

You give an input `.asm` and you get this along with that, so this is what this is the Interfile, I said `myprog.interrite`.

(Refer Slide Time: 9:24)

Module 5.4

Pass 1: Assembler

Initialize Symbol Table

Every line: Remove Comments (Remove whitespace)

Find type of command

myprog.asm  
myprog.inter  
myprog.o

$D = A + M$

$D = A + M$  // Updating Memory (Loop?)

Module 5.4: Assembler: Pass 1

PROF. V. KAMAROTTI  
IIT Madras

That interfile is nothing but this myprog.inter that we are seeing here is nothing but all your comments removed and this plus in addition your entire symbol table is filled all the symbols that are used in that program has an associated address. So this is Pass 1 of your assembler and very quickly I will go through the code, so that you get an understanding.

(Refer Slide Time: 10:05)

```
/****** Beginning of The Final stage (The main() function) */  
//The main() function takes the name of input assembly file as  
command line  
  
void main(int argc, char **argv) {  
    struct SymbolTableEntry item;  
  
    //PASS 1  
    FILE *inputFile;  
  
    //PASS2-Start  
    FILE *interFile;
```

Module 5.4: Assembler: Pass 1

PROF. V. KAMAROTTI  
IIT Madras

So the one of the important thing is we said, right? As we see here I want myprog assembler, myprog.asm, so there is the name of the file is basically given as an argument in your command line, so that were some very interesting C construct, I hope any of you know that, there is something called argc argv if you don't know this it's very important that you know

this, so please go to one of our Google sites or Wikipedia and you will find a lot of tutorials on this.

So this is the way you accept a command line parameter in the C programming.

(Refer Slide Time: 10:59)

```
struct SymbolTableEntry item;

//PASS 1
FILE *inputFile;

//PASS2-Start
FILE *interFile;
//PASS2-End

FILE *outputFile;

int i,j,k; //Temporary variables

int instructionNumber = 0; //This keeps the count of
instructions
int lineNumber = 0; //This keeps the count of number of
lines read

//The nextCommand string will store the command to be
processed
// This is called a buffer
```

NPTEL

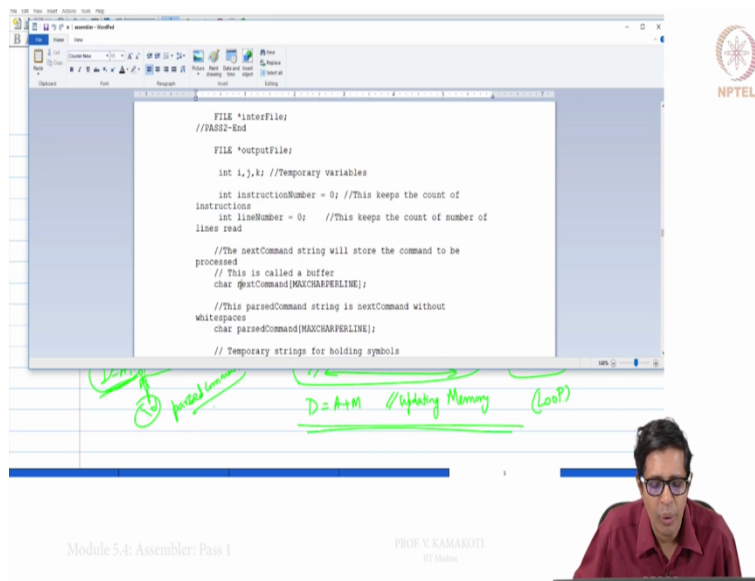
Module 5.4: Assembler: Pass 1

PROF. V. KAMAROTTI  
IIT Madras

Now then you go and you initialize, so one of the thing that we need very quickly as you see here, we need to have an instruction number, we need to have a line number, you have already seen why we need an instruction number, so right? In the module 5.2, when we look at a symbol, when I look at a label with parenthesis immediately I have to assign the instruction number plus 1 for that, right?

Then only that's how I saw the symbol, right? So we need to maintain the instruction number and line number here.

(Refer Slide Time: 11:30)



The screenshot shows a code editor window with the following code:

```
FILE *inFile;
//PASS2-End

FILE *outFile;

int i,j,k; //Temporary variables
int instructionNumber = 0; //This keeps the count of
instructions
int lineNumber = 0; //This keeps the count of number of
lines read

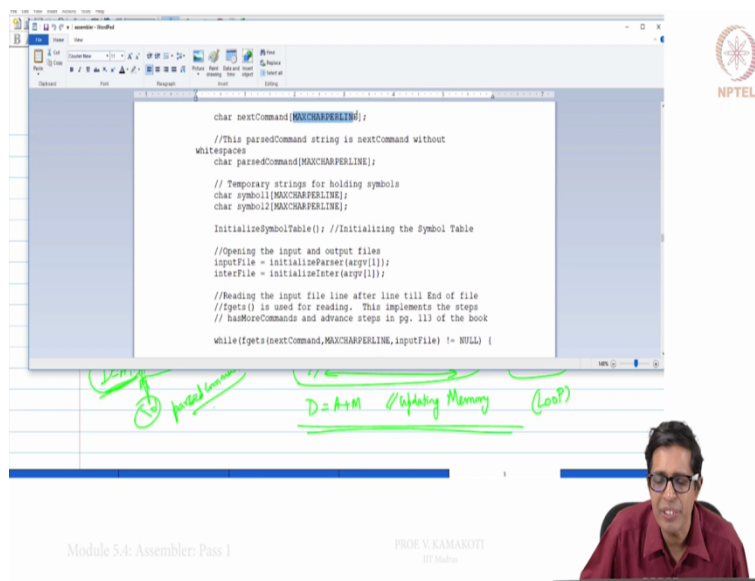
//The nextCommand string will store the command to be
processed
// This is called a buffer
char nextCommand[MAXCHARPERLINE];
//This parsedCommand string is nextCommand without
whitespaces
char parsedCommand[MAXCHARPERLINE];
// Temporary strings for holding symbols
```

Handwritten annotations in green include: "Parsing" with a circled '1', "parsed command", "D = A + M //Updating Memory (Loop)", and "Loop?".

Module 5.4: Assembler: Pass 1  
PROF. V. KAMAKOTTI  
IIT Madras

And what we do here is that, we read one command after command that we call as next command, the past command is nothing but all your comments removed and your whitespace removed your past command is all your whitespace removed here, right?

(Refer Slide Time: 11:47)



The screenshot shows a code editor window with the following code:

```
char nextCommand[MAXCHARPERLINE];
//This parsedCommand string is nextCommand without
whitespaces
char parsedCommand[MAXCHARPERLINE];
// Temporary strings for holding symbols
char symbol1[MAXCHARPERLINE];
char symbol2[MAXCHARPERLINE];

InitializeSymbolTable(); //Initializing the Symbol Table

//Opening the input and output files
inputFile = initializeParser(argv[1]);
interFile = initializeInter(argv[1]);

//Reading the input file line after line till End of file
//fgetc() is used for reading. This implements the steps
// hasMoreCommands and advance steps in pg. 113 of the book
while (fgetc(nextCommand, MAXCHARPERLINE, inputFile) != NULL) {
```

Handwritten annotations in green include: "Parsing" with a circled '1', "parsed command", "D = A + M //Updating Memory (Loop)", and "Loop?".

Module 5.4: Assembler: Pass 1  
PROF. V. KAMAKOTTI  
IIT Madras

So I define these 2 things and you know this is called a buffer, so I have something like 80 characters line, so I put a limit on that. Now I initialize the symbol table, so I put all the 23 entries etc as I...

(Refer Slide Time: 12:06)

```
char symbol1[MAXCHARPERLINE];
char symbol2[MAXCHARPERLINE];

InitializeSymbolTable(); //Initializing the Symbol Table

//Opening the input and output files
inputFile = initializeParser(argv[1]);
interFile = initializeInter(argv[1]);

//Reading the input file line after line till End of file
//fgets() is used for reading. This implements the steps
// hasMoreCommands and advance steps in pg. 113 of the book
while(!getNextCommand(MAXCHARPERLINE, inputFile) != NULL) {
//Inside this loop we parse the command got through
nextCommand
//This is PASS 1 of the assembler and used for only Symbol
// Resolution.
    i = 0;
    j = 0;

```

Module 5.4: Assembler: Pass 1

PROF. V. KAMAKOTI  
IIT Madras

And now I'm going to the most important thing is I'm reading one by one the commands one by one after that and I use this function called fgets to do that.

(Refer Slide Time: 12:18)

```
    i = 0;
    j = 0;
    parsedCommand[0] = '\0'; //Initializing parsed command

    while(nextCommand[i] != '\n') {
//Remove all whitespaces
        if (nextCommand[i] != ' ') && (nextCommand[i] !=
'\t')
            parsedCommand[j++] =
nextCommand[i];
        i++;
    }
    parsedCommand[j] = '\n'; //Add the end of line
    parsedCommand[j+1] = '\0'; //Add the end of string
//Remove comments if any inside a line
    i = 0;
    while (parsedCommand[i] != '\n') {

```

Module 5.4: Assembler: Pass 1

PROF. V. KAMAKOTI  
IIT Madras

And in this part of the code I'm removing all the white spaces and I am also removing all the, if there are comments inside a line I'm using this while loop to remove all the comments, so the and then I'm finding the comment type, it can be an L command or L and A command or L and C command, right?



(Refer Slide Time: 12:43)

```
//Find the commandType and extract symbols
if (parsedCommand[0] != '\n') //This is an instruction
{
    i = commandType(parsedCommand);
    if ((i == L_COMMAND) || (i == L_AND_A_COMMAND) || (i == L_AND_C_COMMAND))
    {
        j = 0;
        if ((parsedCommand[1] < '0') || (parsedCommand[1] > '9'))
        {
            while((parsedCommand[j+1] != '\n') && (parsedCommand[j+1] != '\0')) { //Extracts the first symbol in case of L, L_AND_A and L_AND_C
                symbol[j] = parsedCommand[j+1];
                j++;
            }
            symbol[j] = '\0';
        }
    }
}
```

Module 5.4: Assembler: Pass 1

PROF. V. KAMAROTTI  
IIT Madras

NPTEL

And then what I do is, if it is an L command or any of these commands I basically extract the symbol and I try and add into the symbol table, right? I add that symbol into the symbol table.

(Refer Slide Time: 13:02)

```
Definition for %s in Line %d with command%
\n", symbol, lineNumber+1, parsedCommand);
    exit(-1);
}

//Add the symbol to symbol table
strcpy(item.symbol, symbol);
item.address = instructionNumber;
addEntry(item);
}

instruction
if (i == L_AND_A_COMMAND) {
    // Increment j by 1 to reach the "*" character
    in parsedCommand // Increment j by 1 more to reach the "@"
character in parsedCommand // Increment j by 1 more to reach the first
character of A instruction in parsedCommand
    j = j + 3;
}
```

Module 5.4: Assembler: Pass 1

PROF. V. KAMAROTI  
IIT Madras

And similarly if it's L and commands then I basically have to you know after the symbol after the... So L and A command will look like one symbol with parenthesis and then immediately you're a command will so... So I go to that path and after that @symbol I go and find out if this again another symbol, so @I can have a symbol, right?

So I go and find that and that if it is a symbol again I do this add entry into the symbol table.

(Refer Slide Time: 13:43)

```
}

//Handle Symbols in A_COMMAND
if (i == A_COMMAND)
{
    j = 0;
    if ((parsedCommand[j] < '0') ||
(parsedCommand[j] > '9'))
    {
        while((parsedCommand[j+1] != '\0') &&
(parsedCommand[j+1] != '\n')) {
            //Extracts the first
symbol in case of L, L_AND_A and L_AND_C
            symbol[j] = parsedCommand[j+1];
            j++;
        }
        symbol[j] = '\0';
        //Add the symbol to symbol table
        strcpy(item.symbol, symbol);
        item.address = -1;
        addEntry(item);
    }
}
```

Module 5.4: Assembler: Pass 1

PROF. V. KAMAROTI  
IIT Madras

The last is that if it is just an A command, right? If it is just an A command I just basically see after the @symbol is the character between 0 and 9 if it is not between 0 and 9 then it is a symbol, then again I do the same thing as you have seen, right?

(Refer Slide Time: 14:15)

```
    }
    lineNumber++;
}
// It is not advisable to use
// fgets() in uncontrolled environment as it does not check
// for the buffer bounds. This can cause a vulnerability
// called
// Buffer overflow
//Assign Address to Data variables starting from 16. These
//are symbols with address -1
j = 16;
for (i = 0; i < MAX_ENTRIES; i++)
    if ((strcmp(SymbolTable[i].symbol, "\0") != 0) &&
        (SymbolTable[i].address == -1))
        SymbolTable[i].address = j++;
}
}
```

Handwritten annotations in green:

- Under `j = 16;`: `j = 16;`
- Under the `for` loop: `D = A + M //Updating Memory (Loop?)`

NPTEL logo in the top right corner.

Module 5.4: Assembler: Pass 1  
PROF. V. KAMAROTTI  
IIT Madras

So after finishing all these things this is very very important that I scan every entry of mine Max entries of my symbol table and if I find that one of the address is minus 1 but my entry is /0 is not null, it is not null. So I have a valid string null is but the entry is minus 1 this as I have explained earlier could happen in the case of what? In the case of symbol data points, right?

Like @ I, @ which are non-symbols one, so those things I started assigning address as starting from 16, right? And I keep on implementing. So this is end of Pass 1, so where I close the input file and the interfile. Now I have got an interfile what I write into the interfile is the completely de commented de whitespace thing, right? And this is Pass 1 of the code.

So in the next module we will see how we are going to go about Pass 2.