Computer Organization

605.204

Module Four

Part Two

The Assembler

Module Four

- Part Two
- In this presentation, we are going to talk about :
- Assembler
  - Functions
  - Organization
  - Algorithm

Previously

- Previously we talked about:
- Assembly Language review
- Now: Assembler Functions, Organization, and Algorithm

Assembler

- System Software program
- Translator
- Reads Assembly (People) Language
  - Processes each instruction one line one at a time
- Writes Machine Language

Basic Functions

- **Assign** memory addresses to symbolic labels
- **Translate** mnemonic operation codes to machine operations
- **Convert** data constants into machine codes
- **Build** machine instructions in the proper format
- **Write** the object program and the assembly listing files

- Specific Function details are dictated by:
  - Hardware Architecture
  - Language Design Features

ASSIGN - Memory addresses

Read the input file.

Process the instructions and directives, recording their size. Keep a running location total.

As labels are processed, assign the current location value to the label.

Build a table of label names and location values:

The Symbol Table.

The running location total is called the Location Counter.

```
0     main:   addi $t0, $zero, 0
4             addi $s0, $zero, 0
8             addi $t1, $zero, 100

12    loop:   addi $t0, $t0, 1
16            add  $s0, $s0, $t0
20            blt  $t0, $t1, loop
```

SYMBOL TABLE

main	-	0
loop	-	12

TRANSLATE - Operation codes

- **Translate** mnemonic operation codes to machine operations
  - Cross reference the mnemonic code to the machine code.

- Instruction_Mnemonic Op Code Function Code

– ADD	0	32	SUB	0	34
– MUL	0	24	DIV	0	26
– LW	35	x	LB	32	x
– SW	43	x	SB	40	x
– ADDI	8	x	SLTI	10	x
– BEQ	4	x	BNE	5	x

CONVERT - Data constants

- **Convert** data constants into machine codes
  - Process the BYTE and WORD directives.
  - .byte 0x84 - generate HEX value; one byte
    0x84 = $1000\ 0100_2$
  - .ascii “STOP” - generate ASCII characters - S T O P
    0x53 54 4F 50
  - .word 243 - generate the integer value - 243
    0xF3 = $1111\ 0011_2$

BUILD - Machine instructions

- **Build** machine instructions in the proper format
  - Read operation code from cross reference table
  - Set flag values
  - Insert operand address value

- **`addi $t1, $zero, 100`**
- `Opcode = 8;  rs = 0;  rt = 9;  imm = 100`

- **`001000  00000  01001  0000 0000 01100100`**

MIPS instructions

- R format - three Register operands
- Arithmetic and logical instructions
- **and** $t1, $s3, $s5
- **and** - opcode = 0; function code = 36

op	rs	rt	rd	shamt	funct
000000	10011	10101	01001	00000	100100

MIPS instructions

- I format - Immediate
- Address, branch and immediate arithmetic instructions
- **sw** $t1, 62 ($t2)
- **sw** - opcode = 43

op	rs	rt	16 bit address
101011	01010	01001	0000000000111110

MIPS instructions

- J format - Jump
- Jump, Jump and Link instructions
- **jal** getNext
- **jal** - opcode = 3; getNext = 1316

op	26 bit address

000011	00000000000000000101001001

WRITE - Object program, listing

- **Write** the object program and the assembly listing files
  - Collect the assembled instructions.
  - Insert them into the proper record format and append the records to the object file.
  - Construct the listing file for the programmer.

The Program Organization

- Symbol Table - **SYMTAB**
  - Contains labels and the assigned address
  - Dynamic. Built from empty for each assembly
- Operation Code Table - **OPTAB**
  - Contains all valid instruction mnemonics and corresponding instruction bit pattern (opcode)
  - Static. Built once, reread for each assembly
- Location Counter - **LOCCTR**
  - Single value
  - Running total of program size

The Organization

- Symbol Table - **SYMTAB**
  - Contains labels and the assigned address
  - Dynamic. Built from empty for each assembly

- Name Address Information object type, Cross reference
  - BEGIN 12AF I 12AF
  - LENGTH 342D D 342D, 12AF
  - EXIT 0000 U 5791
  - RDREC 0000 X 1357, 2468

The Organization

- Operation Code Table - **OPTAB**
  - Static. Built once, reread, reused for each assembly
  - (see the Green Card)

- Instruction_Mnemonic Op Code Function Code

–	ADD	0	32		SUB	0	34
–	MUL	0	24		DIV	0	26
–	LW	35	x		LB	32	x
–	SW	43	x		SB	40	x
–	ADDI	8	x		SLTI	10	x
–	BEQ	4	x		BNE	5	x

The Algorithm

- Divide the work into two phases.
- PASS ONE
  - Read each line of the input file.
  - Assign an address to each line.
  - Process the instructions and directives to build the Symbol Table.
- PASS TWO
  - Read each line of the input file.
  - Using the Symbol table, and the Operation Code table build the code for the instruction.
  - Generate constant machine values.
  - Append the code to the object program file.
  - Write the listing file.

The Algorithm

- Why divide the work into two phases ?
- This is the problem: forward reference
  - Symbols (labels) are often referenced before they are defined.
  - For example:
  -

```
                .text
                LW          $s1, Retail ($zero)
                MUL         $t1, $s1, $s7     # price * quantity

                . data
Retail          .float      41.79             # price
```

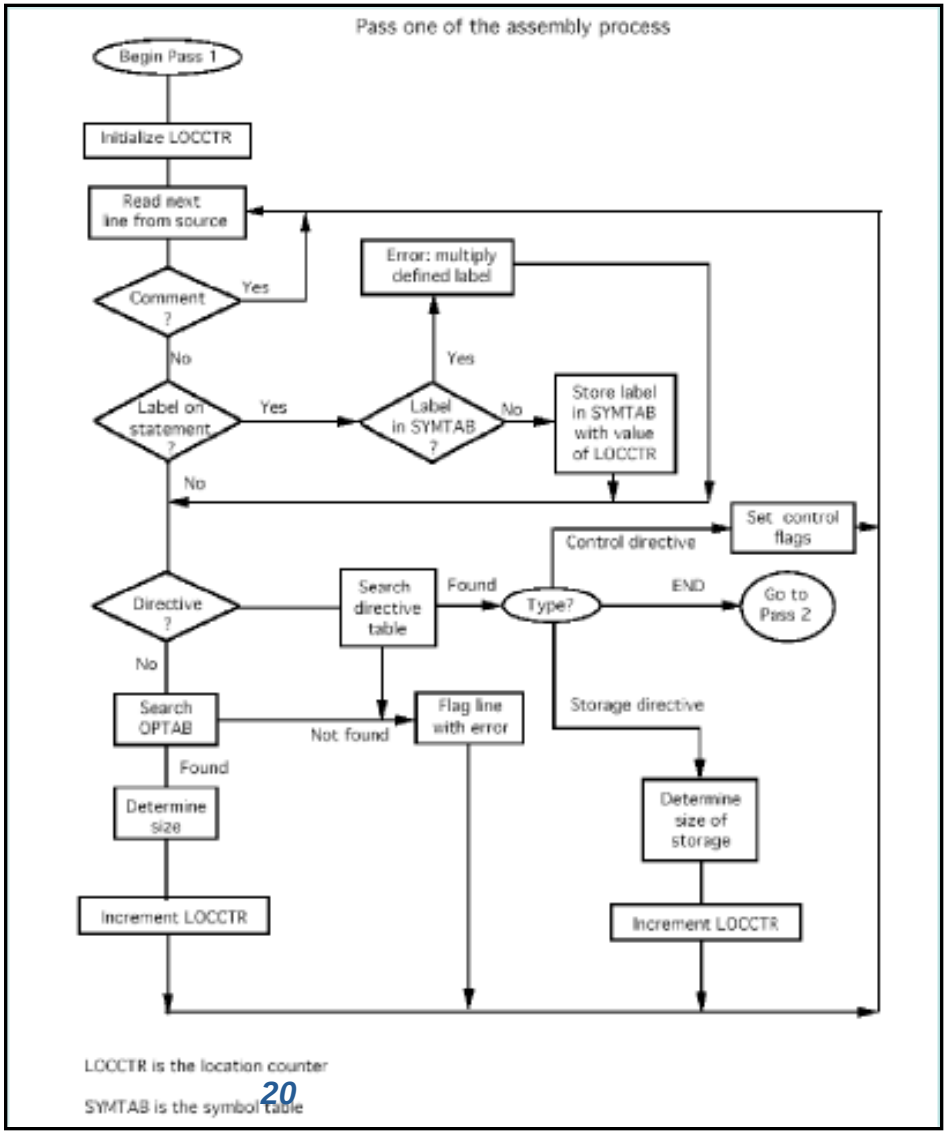
The Algorithm

- PASS ONE - Build the Symbol Table

- Read input line
- Check for ‘START’
- Initialize Location Counter
- Read next input line
  - If it has a label, insert label into symbol table.
  - Verify operation code, increment location counter.
  - If contains directive, increment location counter appropriately.
  - If contains ‘END’ wrap-up.
  - Read again

Pass One

- Physical details need to be added

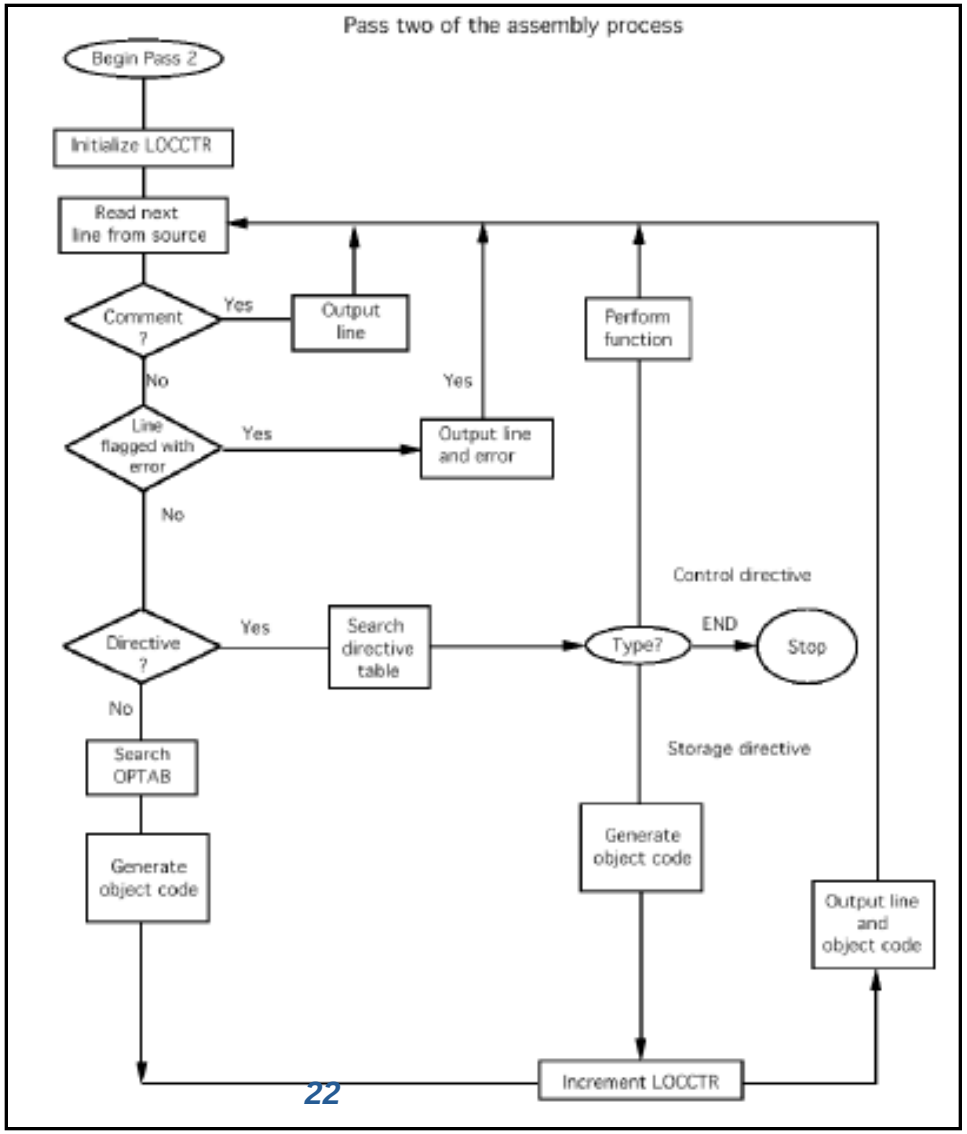


The Algorithm

- PASS TWO - Generate the instructions and object program
- Read input line
- Check for ‘START’
- Initialize the Object program file
- Read next input line
  - Using the Symbol table, and Operation Code table generate the code for this instruction.
  - Generate the constant machine values.
  - Append the instruction code to the object program file.
  - Write the listing file report line.
  - If contains ‘END’ directive, wrap-up Object file. Else, read again.

Pass Two

- Physical details need to be added



Summary

- Assembler
  - Functions
  - Organization
  - Algorithm

- Next: Object File - the Assembler's output file