Patt-Ch7 Assembly Language

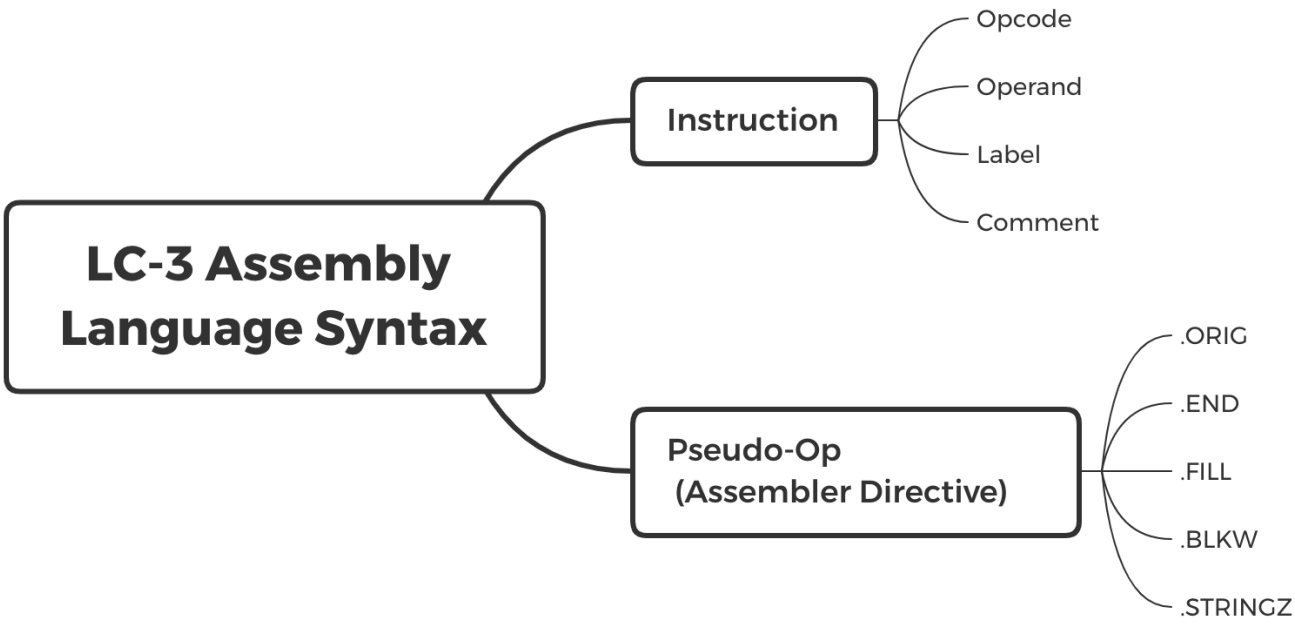
7.1 From Machine Code to Assembly Language

Assembler is a program that turns symbols into machine instructions.

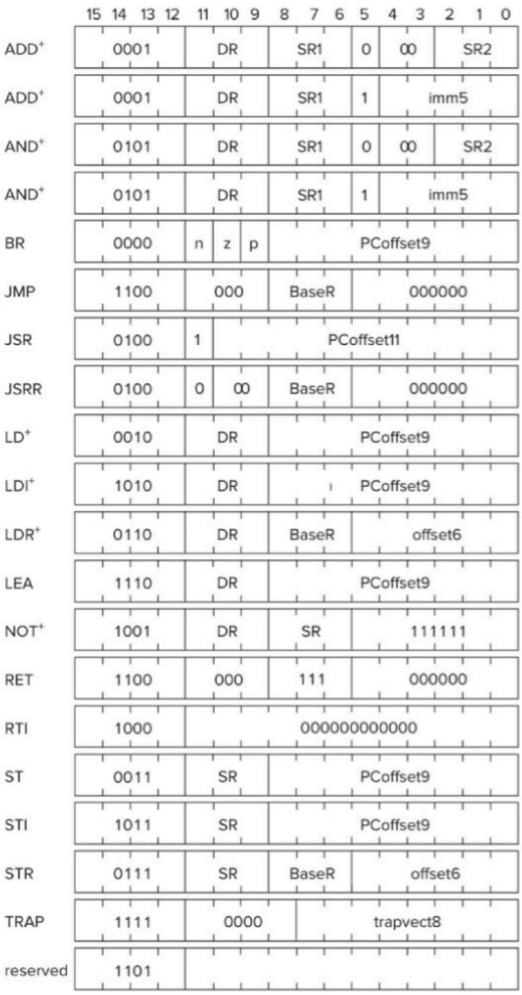
Assembly Language

- Low-level language, ISA-specific
  - it is usually the case that **each ISA has only one assembly language**
- User-Friendly
  - mnemonics(助记符) for opcodes
  - **labels** for memory locations
- still provide detailed control over the instructions

7.2 LC-3 Assembly Language Syntax


LC-3 Assembly Language Syntax
Instruction
Opcode
Operand
Label
Comment
Pseudo-Op (Assembler Directive)
.ORIG
.END
.FILL
.BLKW
.STRINGZ

Machine Code



Assemble Language

ADD DR, SR1, SR2 — ADD r1, r2, r3

ADD DR, SR1, imm5 — ADD r1, r2, #10 (十进制) / b1010 (二进制) / xA (十六进制); [-16, 15]

AND DR, SR1, SR2

AND DR, SR1, imm5

BRn LABEL — BR / BRnzp, BRz, BRp, BRnz, BRnp, BRzp

JMP BaseR

JSR LABEL

JSRR BaseR

LD DR, LABEL

LDI DR, LABEL

LDR DR, BaseR, offset6

LEA DR, LABEL

NOT DR, SR

RET

RTI

ST SR, LABEL

STI SR, LABEL

STR SR, BaseR, offset6

TRAP trapvect8 — TRAP x25

Pseudo-Ops:

.ORIG x3000

.FILL xFFFF

.END

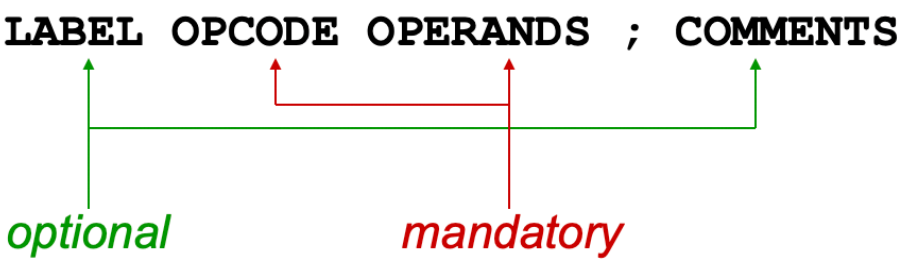
.BLKW 2

.STRINGZ "Hello World"

.EXTERNAL START of FILE *

```
LDR    R4, R2, #-5          ; R4 <- mem[R2-5]
LD     R4, VALUE            ; R4 <- mem[VALUE]
LDI    R4, VALUE            ; R4 <- mem[mem[VALUE]]
LEA    R4, TARGET           ; R4 <- the address of TARGET
ST     R4, HERE             ; mem[HERE] <- R4
STI    R4, THERE            ; mem[mem[THERE]] <- R4
STR    R4, R2, #-5          ; mem[R2-5] <- R4
```

7.2.1 Instructions



7.2.1.1 Opcodes & Operands

Opcodes

- *reserved words*
- Symbolic names

Operands

Operands

- **registers -- specified by Rn, where n is the register number**
- **numbers -- indicated by # (decimal) or x (hex) or b(binary)**
- **label -- symbolic name of memory location**
- **separated by ,**
- **number, order, and type correspond to instruction format**
  - **ex:**

```
ADD  R1,R1,R3
ADD  R1,R1,#3
LD   R6,NUMBER
BRz  LOOP
```

7.2.1.2 Labels

- **placed at the beginning of the line**
- **assigns a symbolic name to the address corresponding to line**
  - **ex:**

```
LOOP  ADD  R1,R1,#-1
      BRp  LOOP
```

2 reasons for using Labels:

- The location is the target of BR
- The location contains a value that is loaded or stored.

7.2.1.3 Comments

- **anything after a semicolon is a comment**
- **ignored by assembler**
- **used by humans to document/understand programs**
- **tips for useful comments:**
  - ➢ **avoid restating the obvious, as “decrement R1”**
  - ➢ **provide additional insight, as in “accumulate product in R6”**
  - ➢ **use comments to separate pieces of program**

7.2.2 Pseudo-Ops (Assembler Directives)

> Pseudo-, /ˈsudoʊ/, 假的

```
.ORIG x3000                ; The start address of program

.END                       ; The end of program

.FILL x0030                ; allocate one word, initialize with value n

.BLKW n(decimal)           ; BLocK of Words, allocate n words of storage

.STRINGZ "ABC"             ; allocate n+1 addresses, initialize n words
                           ; with Zero-EXT ASCII codes, and NULL terminated

.EXTERNAL                  ; send a message to LC-3 assembler that the
                           ; absence of the label is in another module.
                           ; Used for multi-file programming.
```

Note : .END does not stop the program **&** it does not even exist at the time of execution.

7.2.3 Good Programming Style*

Style Guidelines

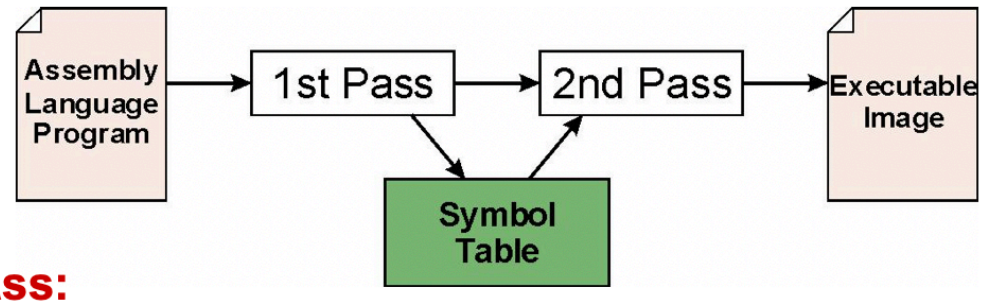
Use the following style guidelines to improve the readability and understandability of your programs:

1. **Provide a program header, with author's name, date, etc., and purpose of program.**
2. **Start labels, opcode, operands, and comments in same column for each line.** (Unless entire line is a comment.)
3. **Use comments to explain what each register does.**
4. **Give explanatory comment for most instructions.**
5. **Use meaningful symbolic names.**
   - **Mixed upper and lower case for readability.**
   - ASCIItoBinary, InputRoutine, SaveR1
6. **Provide comments between program sections.**
7. **Each line must fit on the page -- no wraparound or truncations.**
   - **Long statements split in aesthetically pleasing manner.**

7.3 The Assembly Process

Assembly Process

Convert assembly language file (.asm) into an executable file (.obj) for the LC-3 simulator.



First Pass:

- **scan program file**
- **find all labels and calculate the corresponding addresses; this is called the *symbol table***

Second Pass:

- **convert instructions to machine language, using information from symbol table**

7.3.1 First Pass: Constructing the Symbol Table

1. **Find the `.ORIG` statement, which tells us the address of the first instruction.**
   - **Initialize location counter (LC), which keeps track of the current instruction.**

2. **For each non-empty line in the program:**
   a) **If line contains a label, add label and LC to symbol table.**
   b) **Increment LC.**
      – **NOTE: If statement is `.BLKW` or `.STRINGZ`, increment LC by the number of words allocated.**

3. **Stop when `.END` statement is reached.**

NOTE: A line that contains only a comment is considered an empty line.

```
          .ORIG    x301C
          ST       R3, SAVE3
          ST       R2, SAVE2
          AND      R2, R2, #0
TEST      IN
          BRz      TEST
          ADD      R1, R0, #-10
          BRn      FINISH
          ADD      R1, R0, #-15
          NOT      R1, R1
          BRn      FINISH
          HALT
FINISH    ADD      R2, R2, #1
          HALT
SAVE3     .FILL    x0000
SAVE2     .FILL    x0000
          .END
```

简答题 (2 分)

Symbol Address
TEST x301F
FINISH x3027
SAVE3 x3029
SAVE2 x302A

7.3.2 Second Pass: Generating Machine Language

Potential problems:

- **Improper number or type of arguments**
  - ➢ **ex:** `NOT R1,#7` / `ADD R1,R2` / `ADD R3,R3,NUMBER`
- **Immediate argument too large**
  - ➢ **ex:** `ADD R1,R2,#20`
- **Address (associated with label) more than 256 from instruction**
  - ➢ **can't use PC-relative addressing mode**

7.4 Multiple Object Files (Recommend Reading)

Multiple Object Files

An object file is not necessarily a complete program.

- **system-provided library routines**
- **code blocks written by multiple developers**

For LC-3 simulator,
can load multiple object files into memory,
then start executing at a desired address.

- **system routines, such as keyboard input, are loaded automatically**
  - ➢ **loaded into "system memory," below x3000**
  - ➢ **user code should be loaded between x3000 and xFDFF**
- **each object file includes a starting address**
- **be careful not to load overlapping object files**

7.4.1 The Executable Image

When a computer begins execution of a program, the entity being executed is called an *executable image*. The executable image is created from modules often created independently by several different programmers. Each module is translated separately into an object file. We have just gone through the process of performing that translation ourselves by mimicking the LC-3 assembler. Other modules, some written in C perhaps, are translated by the C compiler. Some modules are written by users, and some modules are supplied as library routines by the operating system. Each object file consists of instructions in the ISA of the computer being used, along with its associated data. The final step is to combine (i.e., *link*) all the object modules together into one executable image. During execution of the program, the FETCH, DECODE, … instruction cycle is applied to instructions in the executable image.

Linking and Loading

Loading **is the process of copying an executable image into memory.**

- **more sophisticated loaders are able to *relocate* images to fit into available memory**
- **must readjust branch targets, load/store addresses**

Linking **is the process of resolving symbols between independent object files.**

- **suppose we define a symbol in one module, and want to use it in another**
- **some notation, such as `.EXTERNAL`, is used to tell assembler that a symbol is defined in another module**
- **linker will search symbol tables of other modules to resolve symbols and complete code generation before loading**