Assembly Languages

LC-3 Machine Instructions

```
1  ; Multiplies a number by six.
2  0011 0000 0000 0000     ; Start at 0x3000.
3  0010 001 000000111      ; Set the counter.
4  0010 010 000000111      ; Load the number.
5  0101 011 011 1 00000    ; Clear the product.
6  0001 011 011 0 00 010   ; Add the number.
7  0001 001 001 1 11111    ; Decrement the counter.
8  0000 001 111111101      ; Add six times.
9  0011 011 000000010      ; Store the product
10 1111 0000 00100101      ; Halt.
11 0000 0000 0000 0110     ; Initial counter
12          ?              ; The number
```

LC-3 Assembly

```
1           ; Multiplies a number by six.
2
3           .ORIG x3000
4           LD   R1, SIX        ; Set the counter.
5           LD   R2, NUM        ; Load the number.
6           AND R3, R3, #0      ; Clear the product.
7   LOOP    ADD R3, R3, R2      ; Add the number.
8           ADD R1, R1, #-1     ; Decrement the counter
9           BRp LOOP            ; Add six times.
10          ST   R3, NUM        ; Store the product.
11          HALT
12
13  SIX     .FILL x0006
14  NUM     .BLKW #1
15          .END
```

Assembly Languages

Definition

An **assembly language** defines symbolic, human-readable representations of machine instructions.

- Each assembly instruction represents one machine instruction.

Example

Consider the following LC-3 machine instruction:

```
0001 001 001 1 11111
```

This instruction is written in LC-3 assembly as:

```
ADD R1, R1, #-1
```

Assembly Languages

Definition

An **assembler** is a program that translates assembly into machine code.

- Assembly lines typically consist of one or more of:
  - An instruction
  - A label
  - A directive
  - A macro
  - A comment
  - Whitespace

Example

The LC-3 simulator's text-to-binary converter functions as an assembler for LC-3 assembly.

Assembly Instructions

An LC-3 assembly instruction has the form:

`OPCODE OPERAND, ..., OPERAND`

- Where *OPCODE* is a reserved symbol for an instruction:
  - `ADD`, `LDR`, `JMP`, `BR`...
- And each *OPERAND* is one of:
  - A register, "`R`*N*"
  - A decimal number, "`#`*N*"
  - A hex number, "`x`*N*"
  - A label

Example

Consider the following assembly instruction:

```
ADD R1, R1, #-1
```

"`ADD`" is the opcode; "`R1`", "`R1`", and "`#1`" are operands.

Labels

Definition

A **label** assigns a symbolic name to an address.

- LC-3 labels may consist of 1 to 20 alphanumeric characters.
  - The first character must be a letter.
  - The label cannot be a reserved symbol.
- The assembler can calculate address offsets using labels.

Example

Consider the following assembly to decrement `R1`:

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

Assembler Directives

Definition

An **assembler directive** is an instruction for the assembler that does not translate into any machine instructions.

- Some directives provide information needed by the assembler.
- Some directives encode data rather than instructions.
- LC-3 assembly includes 5 assembler directives.

Example

To indicate that a program begins at memory location `0x3000`:

```
.ORIG x3000
```

- This specifies the initial 16 bits, `0011 0000 0000 0000`.
- This does not translate into any machine instructions.

Assembler Directives

Directive	Purpose
`.ORIG` *ADDRESS*	Indicates the start of the program, stored starting at memory location *ADDRESS*
`.END`	Indicates the end of the program
`.FILL` *VALUE*	Reserves 1 memory location, initialized with the value *VALUE*
`.BLKW` *N*	Reserves the next *N* memory locations
`.STRINGZ` *STRING*	Reserves the next $n+1$ locations, initialized with a null-terminated *STRING* of length n

Macros

Definition

A **macro** is a rule defining how to substitute one fragment of code for another.

- The LC-3 assembler recognizes 6 predefined macros:
  - `HALT`
  - `GETC`
  - `OUT`
  - `IN`
  - `PUTS`
  - `PUTSP`
- The LC-3 assembler does not support user-defined macros.

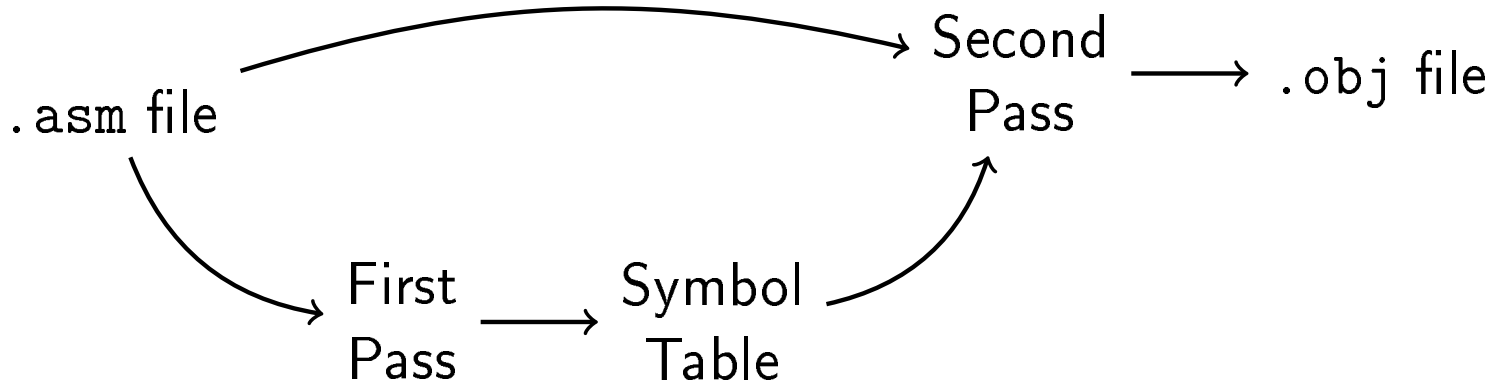
Example

At assembly-time, all occurrences of the macro "`HALT`" are replaced by the actual instruction "`TRAP x25`".

Macros

Macro	Purpose
`HALT`	Request that the instruction cycle be stopped
`GETC`	Request that the ASCII value of one typed character be read into `R0`
`OUT`	Request that the character whose ASCII value is in `R0` be printed to the terminal
`PUTS`	Request that the null-terminated string whose address is in `R0` be printed to the terminal
`IN`	*too specialized to be useful in most programs*
`PUTSP`	

Assembling Programs



- LC-3 assembly files are assembled in two passes.

1. Identify and evaluate labels, storing them in the **symbol table**
2. Translate assembly instructions into machine instructions.

Constructing the Symbol Table

1. Start at the `.ORIG` directive.
2. Set a location counter to the address of the first instruction.
3. For each non-empty, non-comment line, do:
   - If the line begins with a label, add it to the symbol table.
   - If the line is the directive `.BLKW` or `.STRINGZ`, increment the location counter by the number of locations allocated.
   - Else, increment the location counter by 1.
4. Stop when the `.END` directive is encountered.

Constructing the Symbol Table

Example

Consider the program that multiplies a number by six.

- It begins at location `0x3000`.
- It contains three labels: `LOOP`, `SIX`, and `NUM`.

Thus, the symbol table is:

Symbol	Address
`LOOP`	`0x3003`
`SIX`	`0x3008`
`NUM`	`0x3009`

Generating Machine Instructions

1. Start at the `.ORIG` directive.
2. Set a location counter to the address of the first instruction.
3. For each non-empty, non-comment line, do:
   - Replace labels with offsets based on the symbol table.
   - Replace assembly instructions and macros with equivalent machine instructions.
   - Replace `.BLKW`, `.FILL`, and `.STRINGZ` directives with data.
   - Increment the location counter appropriately.
4. Stop when the `.END` directive is encountered.

Generating Machine Instructions

Example

```
        .ORIG  x3000          ; 0011 0000 0000 0000
        LD     R1, SIX        ; 0010 001 000000111
        LD     R2, NUM        ; 0010 010 000000111
        AND    R3, R3, #0     ; 0101 011 011 1 00000
LOOP    ADD    R3, R3, R2     ; 0001 011 011 0 00 010
        ADD    R1, R1, #-1    ; 0001 001 001 1 11111
        BRp    LOOP           ; 0000 001 111111101
        ST     R3, NUM        ; 0011 011 000000010
        HALT                  ; 1111 0000 00100101
                              ;          n/a
SIX     .FILL  x0006          ; 0000 0000 0000 0110
NUM     .BLKW  #1             ; 0000 0000 0000 0000
```

Char Count in Assembly Language (1 of 3)

```
1  ; Program to count occurrences of a char in file.
2  ; Character input from the keyboard.
3  ; Result to be displayed on the monitor.
4  ; Works only if no more than 9 are found.
5  ;
6          .ORIG   x3000
7          AND     R2, R2, #0      ; R2: counter
8          LD      R3, PTR         ; R3: pointer to char
9          GETC                    ; R0 gets char input
10         LDR     R1, R3, #0      ; R1 gets first char
11 ;
12 ; Test character for end of file
13 ;
14 TEST    ADD     R4, R1, #-4     ; Test for EOT
15         BRz     OUTPUT          ; Prepare output
```

Char Count in Assembly Language (2 of 3)

```
1  ; Test character for match, increment count.
2  ;
3          NOT     R1, R1
4          ADD     R1, R1, R0  ; If match, R1 = xFFFF
5          NOT     R1, R1      ; If match, R1 = x0000
6          BRnp    GETCHAR     ; No match
7          ADD     R2, R2, #1
8  ;
9  ; Get next character from file.
10 ;
11 GETCHAR ADD     R3, R3, #1  ; Next character
12         LDR     R1, R3, #0  ; Gets next char
13         BRnzp   TEST
```

Char Count in Assembly Language (3 of 3)

```
1  ; Output the count.
2  ;
3  OUTPUT  LD      R0, ASCII    ; ASCII offset
4          ADD     R0, R0, R2   ; Covert to ASCII
5          OUT                  ; Display ASCII in R0
6          LEA     R0, DoneMsg
7          PUTS
8          HALT                 ; Halt machine
9  ;
10 ; Storage for pointer and ASCII offset
11 ;
12 ASCII    .FILL      x0030
13 DoneMsg  .STRINGZ "Done!"
14 PTR      .FILL      x4000
15         .END
```

Build a Symbol Table

- Fill in the Symbol Table for Char Count

Example

Symbol	Address

Generate Machine Language

- Generate Machine Language for each of the following:

Example

Statement	Machine Language
`LDR R1,R3,#0`	
`ADD R4,R1,#-4`	
`LD R3,PTR`	
`BRnzp TEST`	

Assembler Errors

Examples

What is wrong with each of the following?

- `NOT R1, #7`
- `ADD R1, R2`
- `ADD R3, R3, NUMBER`
- `ADD R1 ,R2, #30`

Assembler Errors

Example

Consider the following assembly instructions:

```
        LD R1, DATA   ; At location 0x3000
                      ⋮
DATA .FILL #1         ; At location 0x3500
```

. . . this fails to assemble; the label is "farther away" than an `LD`'s 9-bit offset can represent.