LC-3 Assembly language

Lecture Topics

LC-3 assembly language

The assembly process

Example

Lecture materials

Textbook Ch. 7

Homework

Machine problem

MP2 due March 2, 2011 at 5pm submitted electronically.

Announcements

LC-3 assembly language

- LC-3 assembly language is a low-level language specifically invented for LC-3 computer
  - It is machine-specific; different processor have different assembly languages
  - Each assembly language instruction corresponds to a single ISA instruction
- Assembly language makes it much easier to program by using human-readable language instead of binary words, while still providing the programmer with the fine-grain control over the instructions and data
- Assembly language allows to refer to *instructions*, *memory locations*, and *values* **symbolically**.

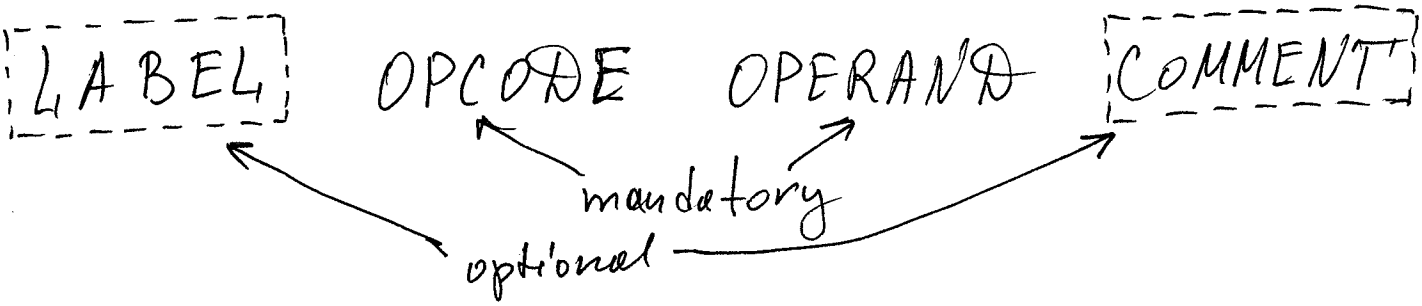
Program from last lecture, but now in LC-3 Assembly language

```
; Program to count occurrences of a character in a file
;
; Initialization
;
    .ORIG       x3000
    AND R2, R2, #0  ; R2 is counter, initially 0
    LD  R3, PTR     ; R3 is pointer to characters
    GETC            ; R0 gets character input
    LDR R1, R3, #0  ; R1 gets first character
;
; Test character for end of file
;
TEST    ADD     R4, R1, #-4 ; Test for EOT (ASCII x04)
    BRz OUTPUT          ; If done, prepare the output
;
; Test character for match.  If a match, increment count.
;
    NOT R1, R1
    ADD R1, R1, R0  ; If match, R1 = xFFFF
    NOT R1, R1      ; If match, R1 = x0000
    BRnp GETCHAR    ; If no match, do not increment
    ADD R2, R2, #1
;
; Get next character from file.
;
GETCHAR ADD     R3, R3, #1  ; Point to next character.
    LDR R1, R3, #0      ; R1 gets next char to test
    BRnzp TEST
;
; Output the count.
;
OUTPUT LD R0, ASCII ; Load the ASCII template
    ADD R0, R0, R2  ; Covert binary count to ASCII
    OUT             ; ASCII code in R0 is displayed.
    HALT            ; Halt machine

;
; Storage for pointer and ASCII template
;
ASCII .FILL x0030
PTR   .FILL x3016
    .END
```

→ Few observations about the program
  → semicolon (;) is used to indicate a comment
  → some lines contain ~~a~~ familiar instructions, e.g., LD, ADD. These are <u>assembly language instructions</u> that will be executed by the computer
  → some lines contain yet unfamiliar instructions, e.g., .FILL, .END. These are <u>pseudo-ops</u>, messages to the translating program to help with the translation to ISA instructions.

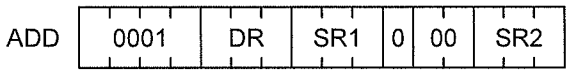
→ An instruction in LC-3's assembly language consists from <u>4</u> parts



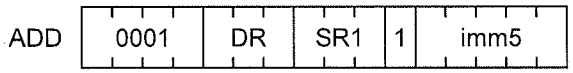
→ <u>OPCODEs</u>
  → are reserved symbols that correspond to LC-3 instructions (it is easier to remember a name than a sequence of 0's and 1's!)
  → 15 opcodes in the LC-3's ISA.

→ <u>OPERANDs</u>
  → each instruction has a number of instruction-specific operands, in specific order
  → operands are one of the following:
    → registers: Rn where n is the register number
    → numbers: # (decimal), x (hex)
    → labels: symbolic names of memory locations
  → operands are seperated by comma (,)

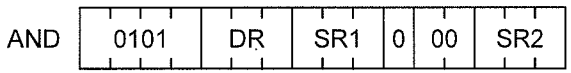
NOTES: RTL corresponds to execution (after fetch!); JSRR not shown



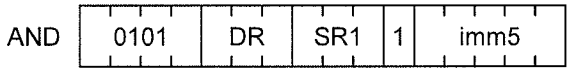
DR ← SR1 + SR2, Setcc



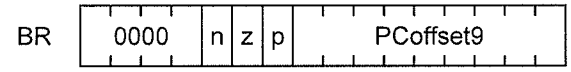
DR ← SR1 + SEXT(imm5), Setcc



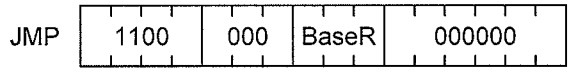
DR ← SR1 AND SR2, Setcc



DR ← SR1 AND SEXT(imm5), Setcc

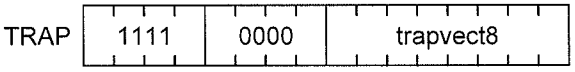


((n AND N) OR (z AND Z) OR (p AND P)):
PC ← PC + SEXT(PCoffset9)

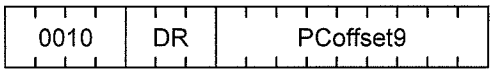


PC ← BaseR

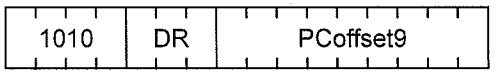
R7 ← PC, PC ← PC + SEXT(PCoffset11)



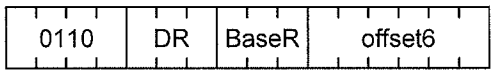
R7 ← PC, PC ← M[ZEXT(trapvect8)]



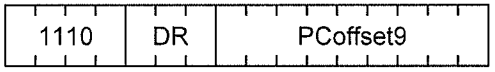
DR ← M[PC + SEXT(PCoffset9)], Setcc



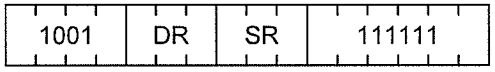
DR ← M[M[PC + SEXT(PCoffset9)]], Setcc



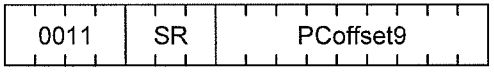
DR ← M[BaseR + SEXT(offset6)], Setcc



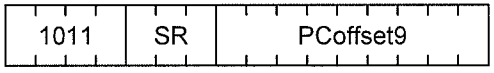
DR ← PC + SEXT(PCoffset9), Setcc



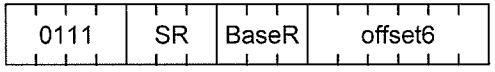
DR ← NOT SR, Setcc



M[PC + SEXT(PCoffset9)] ← SR



M[M[PC + SEXT(PCoffset9)]] ← SR



M[BaseR + SEXT(offset6)] ← SR

→ LABELS

- → symbolic names used to identify memory locations that are explicitly referred to in the program.
- → consist of one-to-20 alphanumeric characters
- → used for 2 reasons
  - → the memory location contains the target of a branch instruction
  - → the memory location contains a value that is loaded or stored by LD-ST family of instructions.
    - → value stored in the memory location explicitly referenced ~~as~~ with the label will be used in the LD/STx instruction.

→ COMMENTS

- → intended only for human consumption
- → their purpose is to make the program more readable by a human
- → ignored by the assembler - more on this later
- → should provide more insight, not just restate the obvious.
- → text after a semicolon (;) is considered to be a comment

→ Pseudo-ops

- → they do not refer to operations that will be actually executed by the computer
- → they really are assembler directives that are used during the assembler language translation into the binary form - more to follow
- → they do not appear in the final binary program.
- → start with dot (.)

→ Pseudo-ops

→ LC-3 assembly language contains 5 pseudo-ops

→ .ORIG specifies where in memory the program should be placed
→ format .ORIG address

→ .END indicates the end of program. It has nothing to do with HALT TRAP, it just says that any characters after it should be ignored
→ format .END

→ .FILL indicates that a location in memory will be filled with a value specified in the operand
→ format .FILL n

→ .BLKW indicates that some number of memory locations will be reserved (block of words)
→ format .BLKW n

→ .STRINGZ indicates that (n+1) memory locations will be reserved and ~~allocated with~~ initialized with the characters provided in the pseudo-instruction.
→ format .STRINGZ "text"
→ example:

```
.ORIG x3010
.STRINGZ "test"
```

⇓

```
x3010  't'
x3011  'e'
x3012  's'
x3013  't'
x3014  0   ← null-terminated string.
```

→ Trap Code pseudo-instructions

- → HALT (TRAP x25)
- → IN (TRAP x23)
- → OUT (TRAP x21)

} you should know these by now

- → GETC (TRAP x20) – read one char from Keyboard
- → PUTS (TRAP x22) – write null-terminated string to console

→ Example #1

Count occurrence of a character in a file.
Character to be input from the Keyboard.
Result to be displayed on the monitor.
Same example as in last lecture

→ Example #2.

Given a sequence of 100 values, stored in 2's complement format starting at x4000, find maximum value. R2 to contain the max value, & R3 to contain the address of the value.

→ Assembly process

→ > lc3as countchar.asm

STARTING PASS 1
∅ errors found in first pass.
STARTING PASS 2
∅ errors found in second pass

> ls
countchar.asm countchar.obj

→ lc3as translated ~~countchar.asm~~ program stored in countchar.asm into a binary code stored in countchar.obj which now can be executed by LC-3 simulator / computer.

→ lc3as is ~~an~~ an LC-3 assembler, a program that translates an assembly language program into a machine language program.

→ 2-pass assembly process

→ 1: creating the symbol table

→ symbol table is ~~a~~ correspondence of symbolic names and their 16-bit memory addresses

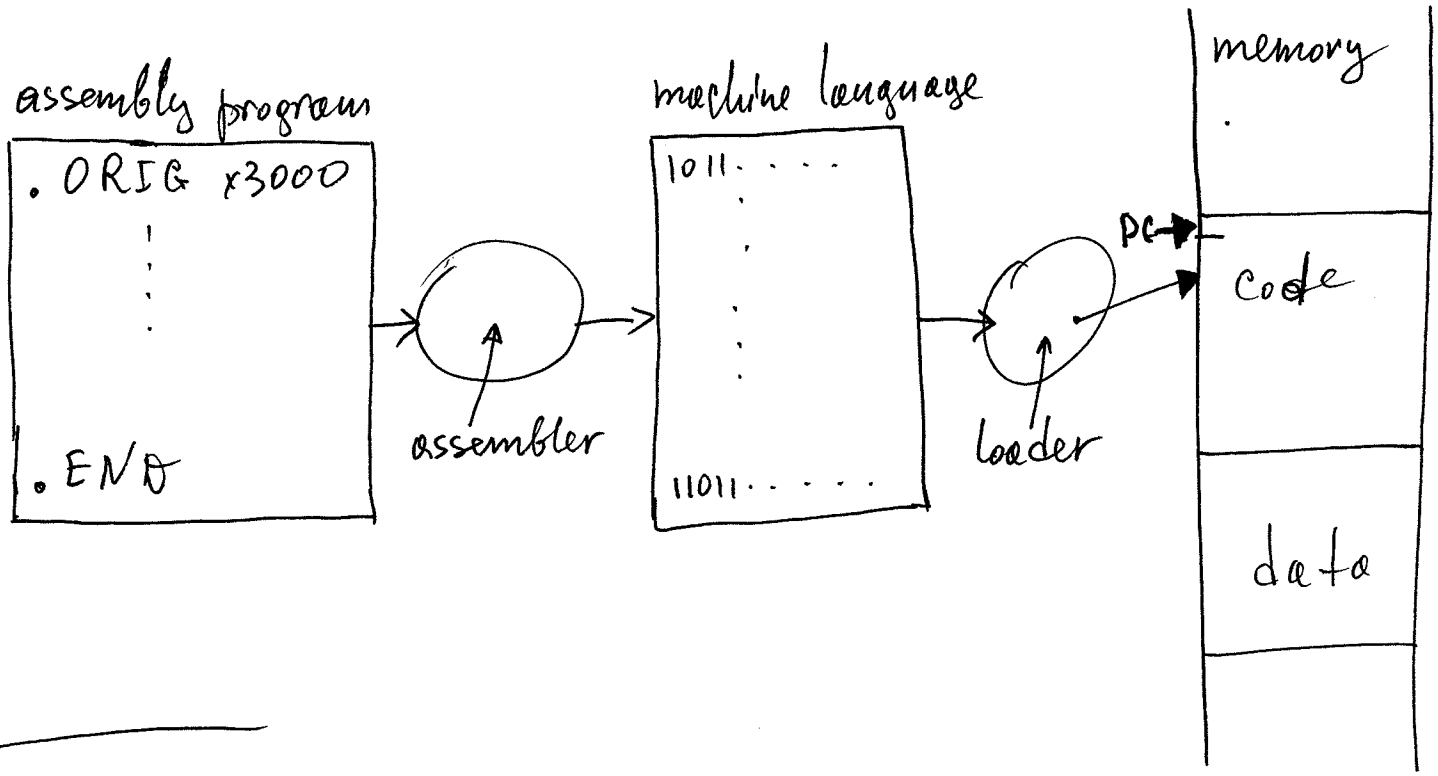
~~symbol address~~
~~PTR x3013~~

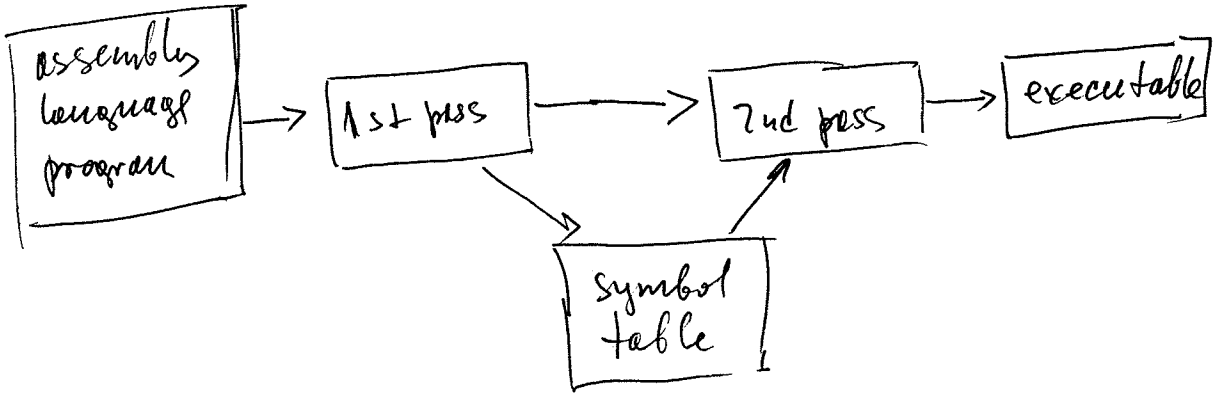
for example #1

symbol	address
TEST	x3004
GETCHAR	x300B
OUTPUT	x300E
ASCII	x3012
PTR	x3013

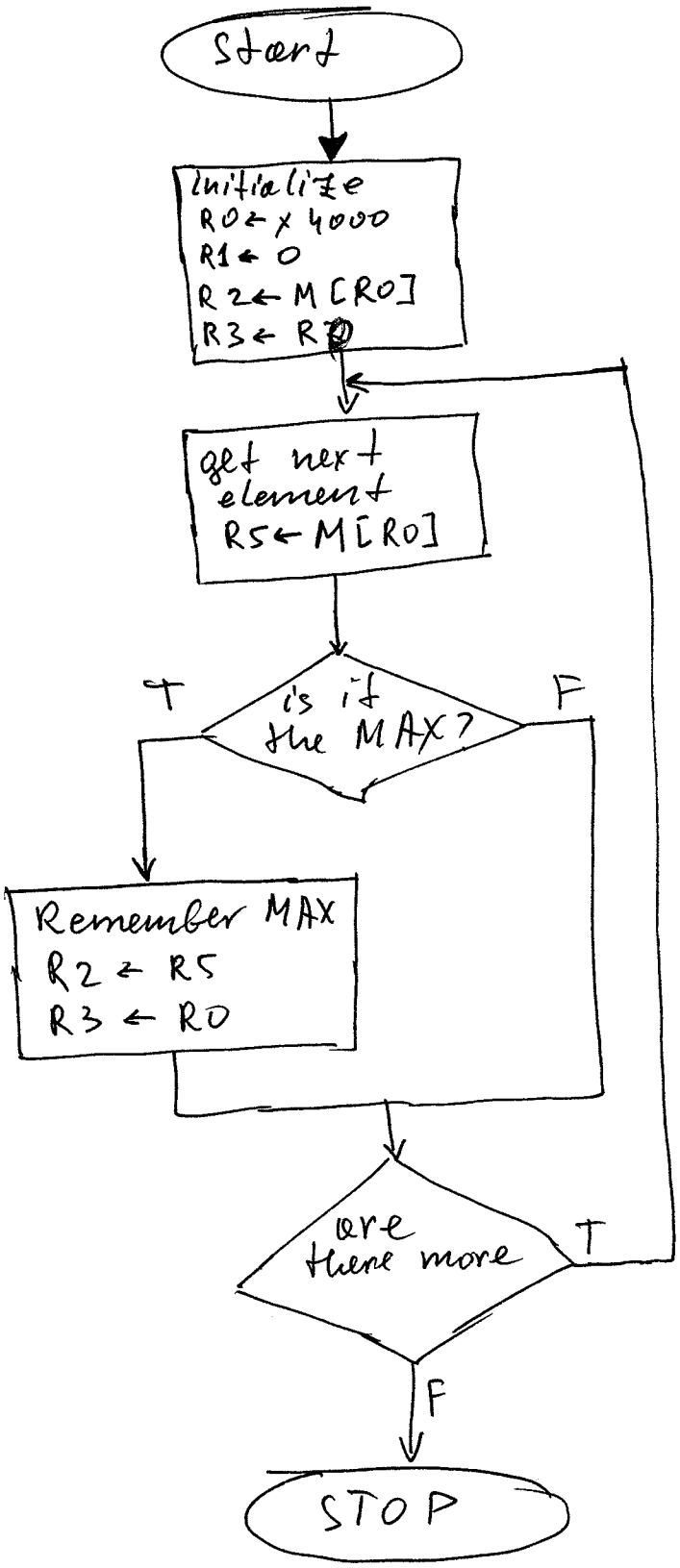
→ each time a new symbol is found, ~~it's~~ the PC value for the line with the symbol is noted.

- 2! generating machine language program
  - → line-by-line, each assembler language instruction is translated into machine (binary) language.
  - → each time a symbol is found as one of instruction's operands, a value from the symbol table is used to resolve the memory address of PC offset.
- from assembler to a running program





Example 2



R0	address of next element
R1	count
R2	current max value
R3	address of current MAX
R4	-100
R5	temp
R6	temp

example 2

```
        .ORIG x3000
        ;
        ; Initialize
        LD R0, ARRAY_START
        AND R1, R1, #0
        LDR R2, R0, #0
        ADD R3, R0, #0
        LD R4, ARRAY_END
        ;
        ; Get next element
LOOP    LDR R5, R0, #0
        ;
        ; is it the max?
        NOT R6, R2
        ADD R6, R6, #1
        ADD R6, R5, R6   ; R6 = R5-R2
        BRnz SKIP
        ;
        ; Remember the Max value
        ADD R2, R5, #0
        ADD R3, R0, #0
        ;
SKIP    ADD R1, R1, #1
        ADD R0, R0, #1
        ;
        ; Are there more?
        ADD R5, R1, R4 ; R5=R1-100
        BRn LOOP
        HALT
ARRAY_START .FILL x4000
ARRAY_END   .FILL #-100
        .END
```

symbol table

Symbol	address
LOOP	x3005
SKIP	x300C
ARRAY_START	x3011
ARRAY_END	x3012