ASSEMBLY LANGUAGE

Michael D'Argenio – mjdargen@ncsu.edu
Electrical Engineering – SS 2019 – Duke TIP

LC-3

Instruction Set Architecture (ISA)

- ISA – all of the programmer-visible components and operations of the computer
- Memory organization
  - address space -- how may locations can be addressed?
  - addressability -- how many bits per location?
- Register set
  - how many? what size? how are they used?
- Instruction set
  - Opcodes
  - data types
  - addressing modes

Instruction Set Architecture (ISA)

- ISA provides all information needed for someone that wants to write a program in machine language.
- Allows you to translate from a high-level language to machine language.
- Or more effectively write programs in high-level language for the target processor by understanding the drawbacks of the ISA.

LC-3 Overview: Memory and Registers

- Memory
  - address space: 2^{16} locations (16-bit addresses)
  - addressability: 16 bits
- Registers
  - temporary storage, accessed in a single machine cycle
    - accessing memory generally takes longer than a single cycle
  - eight general-purpose registers: R0 - R7
    - each 16 bits wide
    - how many bits to uniquely identify a register?
  - other registers
    - not directly addressable, but used by (and affected by)

LC-3 Overview: Instruction Set

- Opcodes
    - 15 opcodes
    - some opcodes set/clear *condition codes*, based on result:
        - N = negative, Z = zero, P = positive (> 0)
- Data Types
    - 16-bit 2's complement integer
- Addressing Modes
    - How is the location of an operand specified?
    - non-memory addresses: *immediate, register*
    - memory addresses: *PC-relative, indirect, base+offset*

LC-3 Opcodes

- *Operate* instructions:
  - ADD
  - AND
  - NOT
- *Data movement* instructions:
  - LD – PC relative mode
  - LDI – indirect mode
  - LDR – base+offset mode
  - LEA - Load effective address -- compute address,save in register
  - ST – PC relative mode
  - STI – indirect mode
  - STR – base+offset mode
- *Control* instructions:
  - BR
  - JSR
  - JMP
  - RTI
  - TRAP

Operate Instructions

- Only three operations: ADD, AND, NOT
- Source and destination operands are registers
  - These instructions *do not* reference memory.
  - ADD and AND can use "immediate" mode, where one operand is hard-wired into the instruction.
- Will show dataflow diagram with each instruction.
  - illustrates *when* and *where* data moves to accomplish the desired operation

Data Movement Instructions

- Load - read data from memory to register
  - LD: PC-relative mode
  - LDR: base+offset mode
  - LDI: indirect mode
- Store - write data from register to memory
  - ST: PC-relative mode
  - STR: base+offset mode
  - STI: indirect mode
- Load effective address - compute address, save in reg
  - LEA: immediate mode
  - does not access memory

Control Instructions

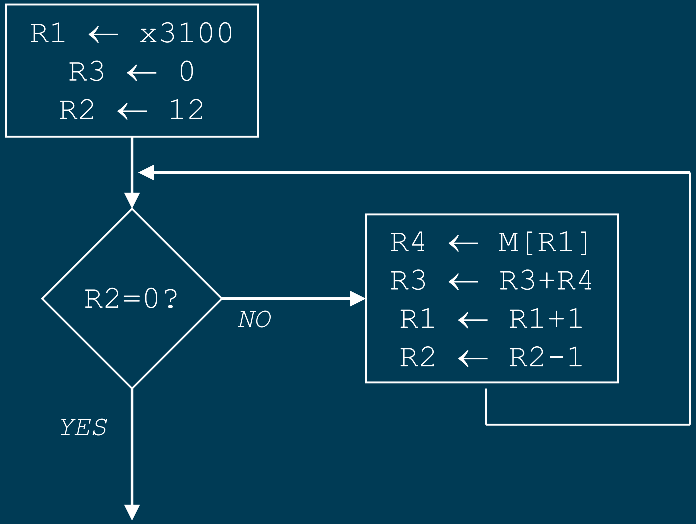
- Used to alter the sequence of instructions (by changing the Program Counter)
- Conditional Branch
  - branch is *taken* if a specified condition is true
    - signed offset is added to PC to yield new PC
  - else, the branch is *not taken*
    - PC is not changed, points to the next sequential instruction
- Unconditional Branch (or Jump) always changes the PC
- TRAP
  - changes PC to the address of an OS “service routine”
  - Returns control to the next instruction(after TRAP

Condition Codes

- LC-3 has three condition code registers:
  - N – negative
  - Z – zero
  - P – positive (greater than zero)
- Set by any instruction that writes a value to a register
  - (ADD, AND, NOT, LD, LDR, LDI, LEA)
- Exactly <u>one</u> will be set at all times
  - Based on the last instruction that altered a register

Example Program

- Compute sum of 12 integers.
  - Numbers start at location x3100.
  - Program starts at location x3000.



Example Program

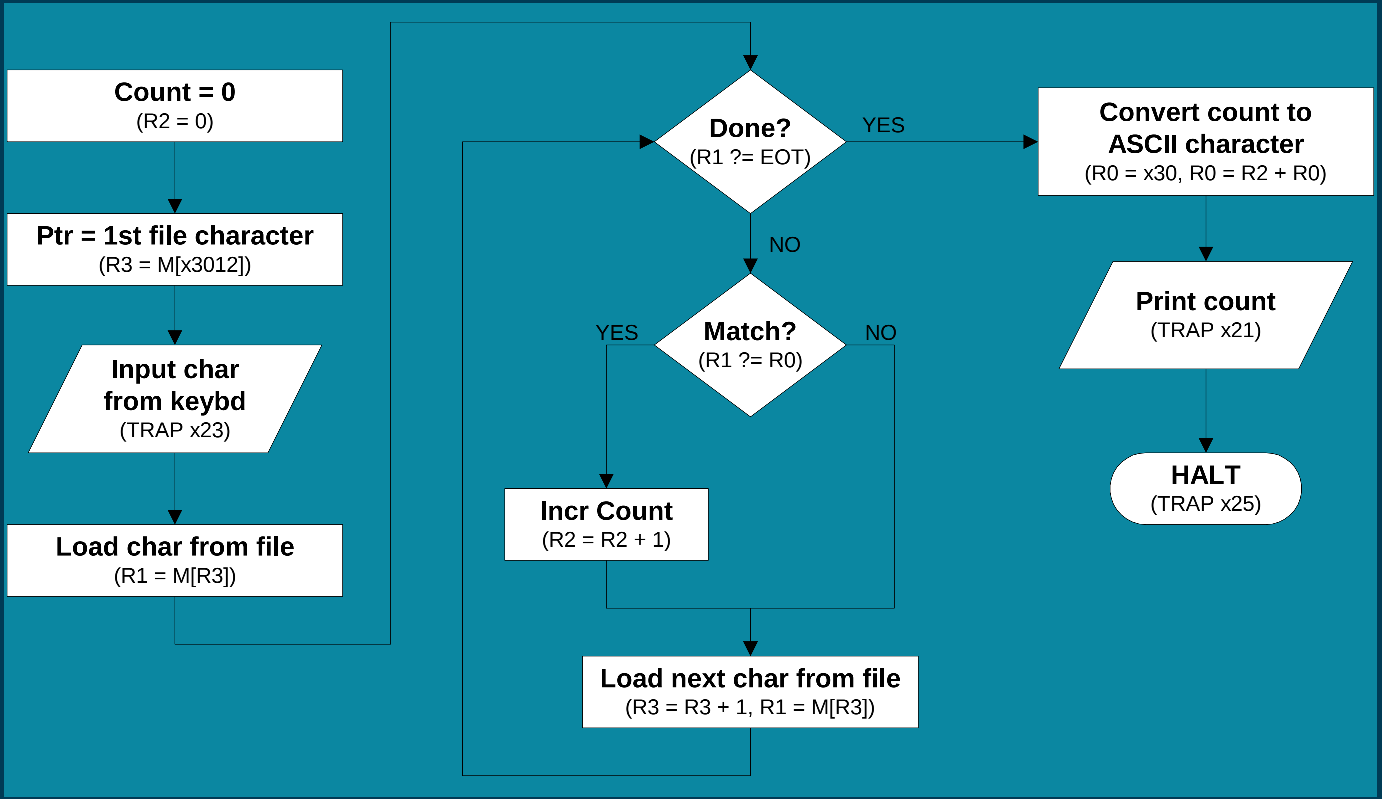
Address	Instruction	Comments
x3000	1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1	R1 ← x3100 (PC+0xFF)
x3001	0 1 0 1 0 1 1 0 1 1 1 0 0 0 0 0	R3 ← 0
x3002	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	R2 ← 0
x3003	0 0 0 1 0 1 0 0 1 0 1 0 1 1 0 0	R2 ← 12
x3004	0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1	If Z, goto x300A (PC+5)
x3005	0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0	Load next value to R4
x3006	0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1	Add to R3
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	Increment R1 (pointer)
X3008	0 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1	Decrement R2 (counter)
x3009	0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0	Goto x3004 (PC-6)

Example

Count the occurrences of a character in a file

- Program begins at location x3000
- Read character from keyboard
- Load each character from a “file”
- If file character equals input character, increment counter
- End of file is indicated by a special ASCII value: EOT (x04)
- At the end, print the number of characters and halt.

Flow Chart



Count = 0
(R2 = 0)
Ptr = 1st file character
(R3 = M[x3012])
Input char from keybd
(TRAP x23)
Load char from file
(R1 = M[R3])
Done?
(R1 ?= EOT)
YES
NO
Match?
(R1 ?= R0)
YES
NO
Incr Count
(R2 = R2 + 1)
Load next char from file
(R3 = R3 + 1, R1 = M[R3])
Convert count to ASCII character
(R0 = x30, R0 = R2 + R0)
Print count
(TRAP x21)
HALT
(TRAP x25)

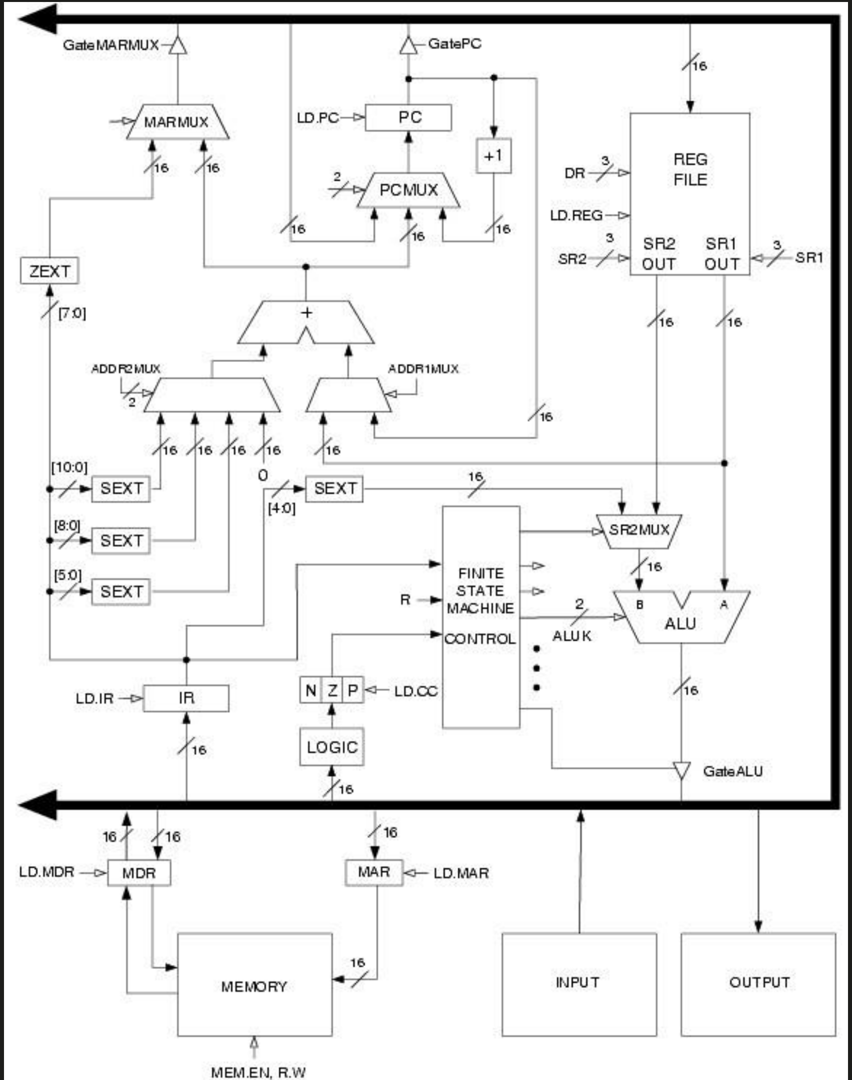
Program (1 of 2)

Address	***Instruction***	***Comments***
x3000	0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0	*R2 ← 0 (counter)*
x3001	0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0	*R3 ← M[x3102] (ptr)*
x3002	1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1	*Input to R0 (TRAP x23)*
x3003	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	*R1 ← M[R3]*
x3004	0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 0	*R4 ← R1 – 4 (EOT)*
x3005	0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0	*If Z, goto x300E*
x3006	1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1	*R1 ← NOT R1*
x3007	0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1	*R1 ← R1 + 1*
X3008	0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0	*R1 ← R1 + R0*
x3009	0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1	*If N or P, goto x300B*

Program (2 of 2)

Address	***Instruction***	***Comments***
x300A	0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 1	*R2 ← R2 + 1*
x300B	0 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1	*R3 ← R3 + 1*
x300C	0 1 1 0 0 0 1 0 1 1 0 0 0 0 0 0	*R1 ← M[R3]*
x300D	0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 0	*Goto x3004*
x300E	0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0	*R0 ← M[x3013]*
x300F	0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0	*R0 ← R0 + R2*
x3010	1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 1	*Print R0 (TRAP x21)*
x3011	1 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1	*HALT (TRAP x25)*
X3012	**Starting Address of File**	
x3013	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	*ASCII x30 ('0')*

DATA PATH


GateMARMUX
GatePC
MARMUX
LD.PC
PC
+1
PCMUX
REG FILE
DR
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
ZEXT
[7:0]
+
ADDR2MUX
ADDR1MUX
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
0
[4:0]
SEXT
SR2MUX
FINITE STATE MACHINE
R
CONTROL
ALUK
ALU
B
A
LD.IR
IR
N
Z
P
LD.CC
LOGIC
GateALU
LD.MDR
MDR
MAR
LD.MAR
MEMORY
MEM.EN, R.W
INPUT
OUTPUT
16
3
2

Data Path Components

- Global bus
  - special wires that carry a 16-bit signal to many components
  - inputs to the bus are "tri-state devices," that only place a signal on the bus when they are enabled
  - only one (16-bit) signal should be enabled at any time
    - control unit decides which signal "drives" the bus
  - any number of components can read the bus
    - register only captures bus data if it is write-enabled by the control unit
- Memory
  - Control and data registers for memory and I/O devices
  - memory: MAR (mem address reg) MDR (mem data reg).

Data Path Components

- ALU
  - Accepts inputs from register file and from sign-extended bits from IR (immediate field).
  - Output goes to bus.
    - used by condition code logic, register file, memory
- Register File
  - Two read addresses (SR1, SR2), one write address (DR)
  - Input from bus
    - result of ALU operation or memory read
  - Two 16-bit outputs
    - used by ALU, PC, memory address
    - data for store instructions passes through ALU

Data Path Components

- PC and PCMUX
  - Three inputs to PC, controlled by PCMUX
    1. PC+1 – FETCH stage
    2. Address adder – BR, JMP
    3. bus – TRAP (discussed later)
- MAR and MARMUX
  - Two inputs to MAR, controlled by MARMUX
    1. Address adder – LD/ST, LDR/STR
    2. Zero-extended IR[7:0] -- TRAP (discussed later)

Data Path Components

- Condition Code Logic
    - Looks at value on bus and generates N, Z, P signals
    - Registers set only when control unit enables them (LD.CC)
- Control Unit – Finite State Machine
    - On each machine cycle, changes control signals for next phase of instruction processing
        - who drives the bus? (GatePC, GateALU, ...)
        - which registers are write enabled? (LD.IR, LD.REG, ...)
        - which operation should ALU perform?
        - ...
    - Logic includes decoder for opcode, etc.