CS 2110 Quiz 3 Notes

Krish Katariya

Last updated: **March 31, 2024**

Contents

1. I/O (Input / Output)

The LC-3 uses **memory-mappings** (reserved locations of its memory) for communicating with registers stored in external devices

Memory mappings in LC3 work by reserving specific locations in its memory for this purpose. Different addresses are dedicated to specific functions, such as handling input from a keyboard. By writing to or reading from these reserved addresses, LC3 can perform I/O operations without specific I/O instructions in the ISA.

1.1. Synchronous vs Asynchronous I/O

Definition 1.1.1

Synchronous IO - data is transferred at a fixed rate both our system and the device can agree on. The CPU reads/writes every x cycles

Asynchronous IO - data is transferred at an unpredictable rate (typical of keyboard input/monitor output).

- In order to synchronize the system and the device, we use **flags** (signals for communicating when the data is ready to be used)

1.2. IO Techniques

Definition 1.2.1

Polling - the *processor* repeatedly checks if data is ready and proceeds when it is **Interrupts** - the *device* interrupts the execution of the processor and demands that a routine is executed instantly with whatever data is supplying

1.3. Keyboard Input

TBD

1.4. Display Output

TBD

2. Operating System of the LC-3

2.1. Privilege

There are two types of privilege in LC-3:

Supervisor mode is the mode that operating system code executes in. This mode carries out I/O routines, and other crucial system behavior.

- In supervisor mode, a program can access all locations in memory and execute all instructions

User mode is the mode that user code executes in. This mode is restricted from accessing some parts of memory such as OS code and I/O space

2.2. Priority

A computer like the LC-3 might need to handle many different kinds of program and events, which is why we ned to assign them a priority.

Theorem 2.2.1

Every program gets given a priority level `PRn`, ranging from `PL0` to `PL7`

Higher numbers have higher priority, so PL7 has the highest priority.

2.3. Process Status Register (PSR)

The status of the currently executing program is stored in the PSR. The status includes information about the privilege and priority of the program, and the current CC codes.

```
PSR[15] == 1 <- User Mode
PSR[15] == 0 <- Supervisor Mode
```

2.4. Memory Layout

Definition 2.4.1

Privileged Memory: (**x0000 - x2FFF**) the part of memory that requires supervisor privileges to access.

- It contains code and data for the operating system, including a supervisor stack

x000 - **Supervisor Stack Pointer**:

This part of memory contains most of the OS, including code ofr system processes such as TRAP system routines

SSP - x2FFF: This part of memory is the supervisor stack, which grows and shrinks as SSP moves.

Definition 2.4.2

User Memory: (x3000 - xF300)

x3000 - **User Stack Pointer**:

This part of memory contains the main function at x3000, and any subroutines the user wrote and any dat needed by the user programs.

USP - xFDFF:

This part of memory is the user stack, it can grow and shrink as the USP moves

Definition 2.4.3

I/O Page - xFE00 to xFFFF

- This part of memory isnt mapped to actual memory, but registers are memory mapped here, meaning that registers are given an address, and using that address accesses that register instead. This can only be accessed using **supervisor privileges**

3. Traps

3.1. Traps Overview

Definition 3.1.1

TRAPs are service routines built into the LC-3 that helps simplify instructions. This is the trap vector format:

```
[1111] [0000] [trapvect8]
```

TRAPs have an opcode (1111), and an **8-bit trap vector**, which allows us to preform different functions.

Theorem 3.1.1

Common Trap Aliases

1. HALT - x25: stops LC3 from running
2. OUT - x21: print character in R0 to console
3. PUTS - x22: print a string of characters, staring at the address in R0 and ending at a null character

The trap vector is actually an address that points to another address of a trap handler

- **trap handlers** are a bundle of instructions that a TRAP oerforms

TRAP Instruction Formula: $\text{PC} \leftarrow \text{MEM}[\text{ZEXT}(\text{TRAPvect8})]$

3.2. Trap Vector Table

As a recap, x0000 - x2FFF is allocated for the system. The **trap vector table** (and the interrupt table) lies from x0000 to x01FF, and serves as a lookup table in memory.

Each reference in the lookup table corresponds to the memory address of the *start* of a trap handler somewhere else in the system memory.

3.3. Interrupts

SOmetimes its not realistic for a current program to totally halt to poll I/O. In this case, we use **interrupt driven I/O**, which allows a program to run normally until the moment that an event comes in.

From that point, the processor immediately handles the appropriate event with a handler code before going back to the original executing program.

3.4. Interrupts Overview

Theorem 3.4.1

External devices can assert interrupts with pauses the current executing of the program. This requires three conditions.

1. The computers currently allows this device to interrupt
2. The decide wants to assert an interrupt
3. The interrupt has higher priority than the program currently executing

KBSD and KBDR have an extra bit de fined as "interrupt enable" bit so that the processor can turn on/off the ability of specific devices to interrupt LC-3.

We use a **priority encoder** to detect the "interrupt signal" from these devices and output the priority level of the highest priority device requesting an interrupt. From there, the highest priority level is compared against the priority level of the current program.

- If the highest interrupt priority level is higher than the current programs, the INT signal at the bottom of the diagram outputs 1, and we interrupt the processor before the next fetch. If not, INT is 0, and we don't interrupt the processor.

Definition 3.4.1

The **interrupt vector table** is a table which maps an 8-bit interrupt vectory to the memory address of the start of a chunk of service routine code somewhere between.

The main differences with this and the trap vector table is that instead of zero extending we append x01 as the higher 8 bits

4. C Compilation

C sources files (or .c files) are typical code files where all our functions are written.

C header files contains functions and global variables to be shared between several source files. In order to use a header file, you can include it using `#include file.h`

4.1. Preprocessor

Definition 4.1.1

The **C preprocessor** performs text-level substitution on C source files and header files before they reach the compile

Theorem 4.1.1

The major **text-level** level substitution are:

1. **#include** (file inclusion)
2. **#define** (macro expansion)
3. **Conditional compilation**
4. **Code line identification**

Definition 4.1.2

File Inclusion

C header files can be included to other header or sources files with #include. The preprocessor will satisfy these inclusions by *literally copying the entire contents of the included file into the other file.*

Definition 4.1.3

Macros

Macros are a *preprocessor directive*, essentially meaning that they tell the preprocessor to do something before compilation. They are just advanced text replacement.

This is how they are defined: `#define MACRO_NAME(ARGUMENTS) TEXT_REPLACEMENT`

An important thing about macros is that they are just text-replacement, and you should **always** surround the entire expression and every use of an argument in parentheses.

So you should write:

```
#define MULTIPLY(X, Y) ((X) * (Y)) (NOT X * Y)
```

4.2. The C Compiler

The C compiler is responsible for taking in the pre processed C code and converting that into machine code. It has two major phases:

1. **Source Code Analysis**: Source code is parsed and broken down into smaller parts
2. **Target Code Synthesis**: Constructing machine code representation for each of the target parts

4.2.1. Symbol Table

If C, we can use function names, variable names, etc. The compiler makes use of a symbol table for all of these.

4.2.2. Compiled Code

Compiled code will be .c translated into binary files, using the ISA of what our target system is. It outputs an object file, which is just machine code for one part of the program.

4.3. The Linker

Object files are not executable on their own, but the linker takes multiple object files and creates one executable file to run your program.

5. C Variables

Definition 5.1

Data Types:

- **int**
- **char**: 8 bits
- **double**: at the lowest level, a floating point number is a bit battery where one of the bits represents the sign of the number, several other bits represent the mantissa, and the remaining represent the exponent

5.1. Identifiers

- Identifiers can be composed of letters from the alphabet, digits, and _. Only letters and the underscore character can be used to start the identifier.
- Cannot be a keyword

- Case-sensitive
- Variable starting with an underscore are usually only in special library code
- Uppercase is reserved for symbolic #define values
  - Can you use it anyways? Syntax or general rule?

5.2. Size Of

The size of a data type depends on the system the program has been compiled to run on. The best way to find this is to use sizeof(int)