CS 2110 Quiz 2 Study Guide

Krish Katariya

Last updated: **February 18, 2024**

Contents

1. K-Maps

1.1. Gray Codes

Definition 1.1.1

Gray Codes are a specific ordering of binary system where successive values can only differ by one bit.

1.2. Step 1: Create the K-Map

Distribute the variables across the rows and columns using Gray Code order, and fill in the corresponding entries with either 1′s, 0′s, or X's

Note 1.2.1

We use **X**'s when we have a "don't care condition". Basically, this is when a certain condition will never occur or are not important to an application. You can choose these to be 0/1 if it makes grouping easier.

1.3. Step 2: Make Groupings

Theorem 1.3.1

Rules for Groupings:
- Groups must be rectangular, but can wrap around edges and corners
- Size of groups must be a power of 2 (including size 1)
- Have as few groups as possible

1.4. Step 3: Create Simplified Expression

You can find the simplified expression based on which terms *don't* change between the 1's in a group

2. Storage

2.1. Address Space & Addressability

Definition 2.1.1

Address Space is the number of distinct locations you can access in memory.

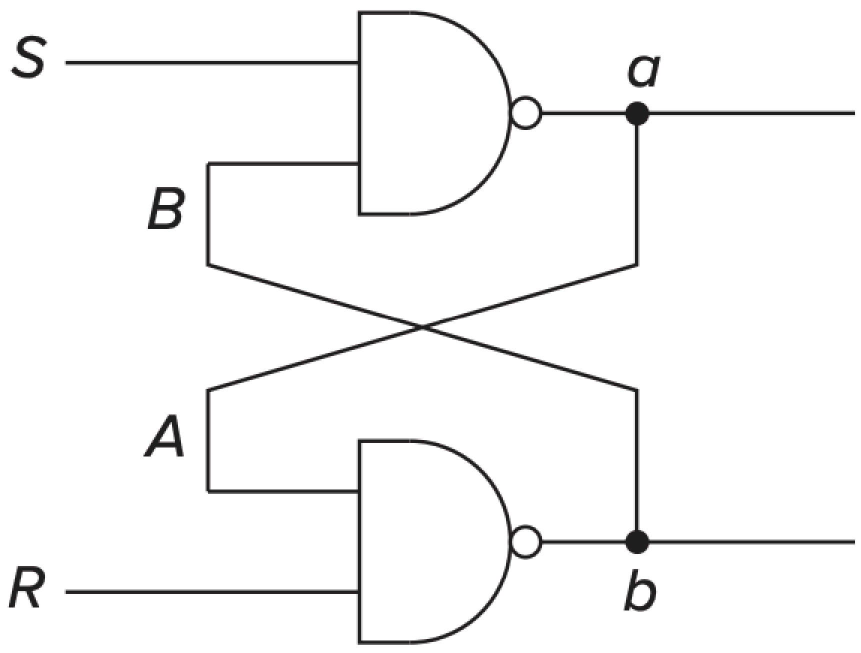
Definition 2.1.2

Addressability is the amount of data stored at each location

`Total Memory = address space * addressability`

2.2. RS Latch

These are the basis of sequential logic, used to remember 1 bit.

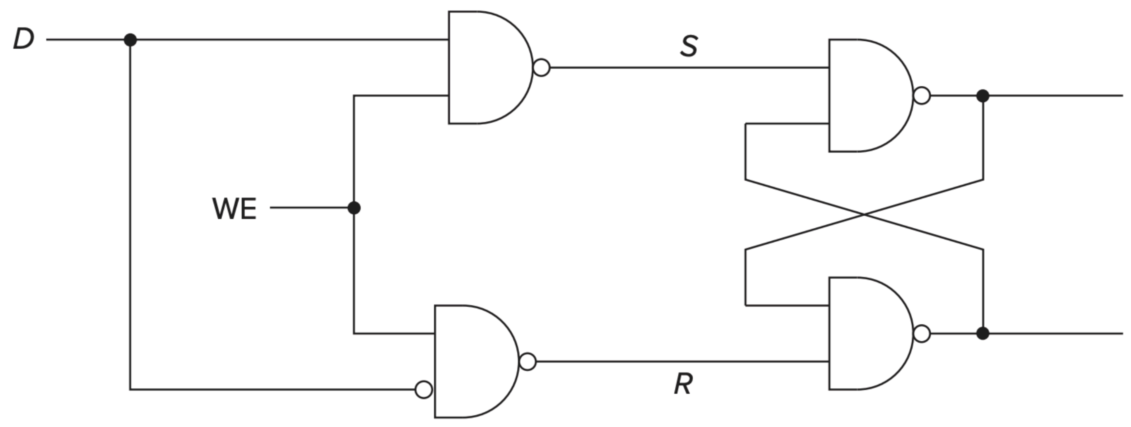


Note 2.2.1

RS Latches are in a **quiescent state** when both S and R are 1. If we set S to 0, it sets the value to 1, and if we set R to 0, it resets the value to 0. Both S and R being 0 is an invalid state that we don't use.

2.3. Gated D-Latch

These are RS Latches with a modified interface that allows us to access the value of storage when D = 1, and flip the value of storage when D = 1 and WE (write enable) = 1.



re 3.19 A gated D latch.

2.4. D Flip Flops

Gated D latches update immediately when WE is 1. Instead with D Flip FLops, the output changes only when WE changes from 0 to 1.

2.5. Edge vs. Level Triggered Logic

Definition 2.5.1

Level Triggered Logic: change happens whenever the clock has a 1 input and is level. So any time its at 1, the output can change. On the other hand, **edge triggered logic** only changes at the exact moment that the clock switches from 0 to 1.

Note 2.5.1

RS Latches, D Latches, and memory are all level triggered.

Flipflops and registers are edge triggered.

Registers are often only use for short term memory access, so we have them be edge triggered for synchronous operation where data is only transferred when the clock signal changes, ensuring its reliable.

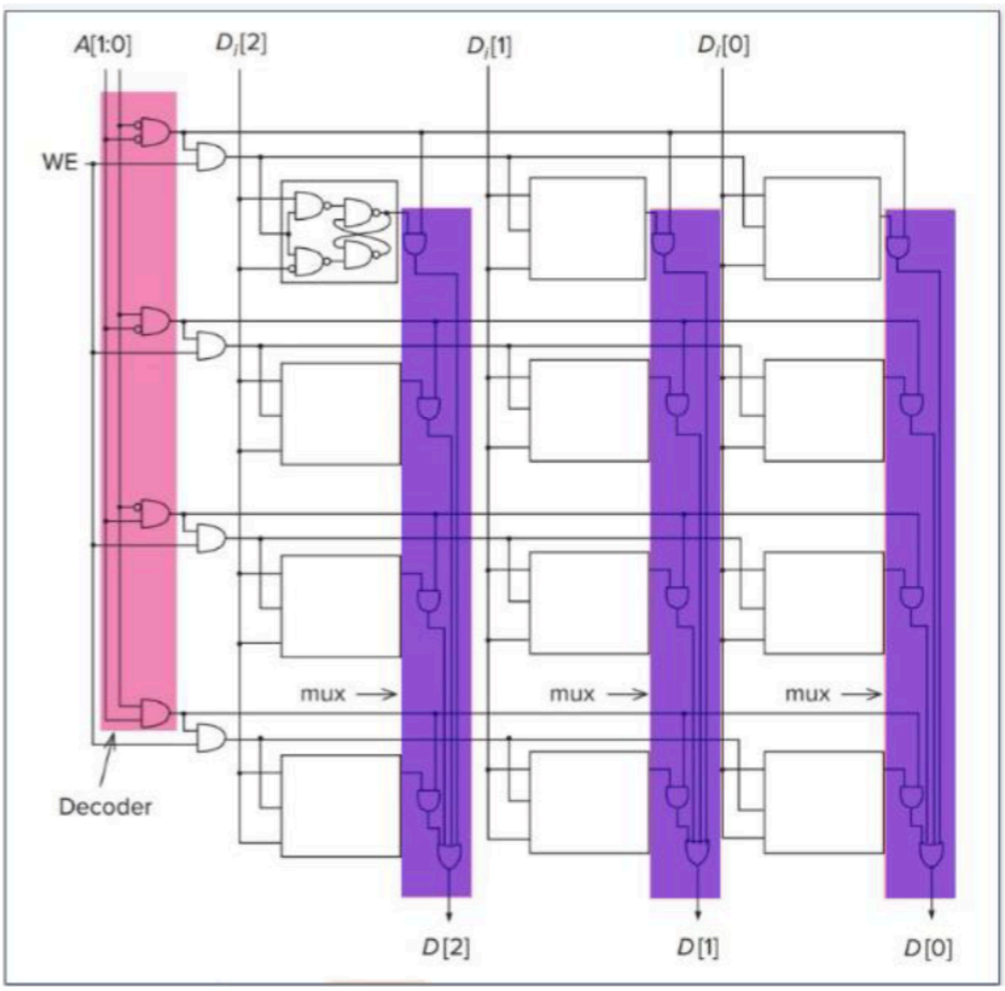
On the other hand, memory is often level triggered because this allows for continuous access to stored data, which allows for asynchronous operation where data can be accessed without strict synchronization.

Registers

Since registers are edge triggered, we created them simply by combining multiple D flip flops

Memory

Memory is more complicated, we use a lot of Gated D Latches alongside a decoder (to find which address line to get)



3. State Machines

Digital logic structures that *both* process information *and* store decisions are called **sequential logic circuits**.

Definition 3.1

The **state** of a system is a snapshot of all the relevant elements of the system at the moment the snapshot is taken.

Definition 3.2

A **finite state machine** consists of five elements

1. A finite number of states
2. A finite number of external inputs
3. A finite number of external outputs
4. An explicit specification of all state transitions
5. An explicit specification of what determines each external output value

State machines take in a state and action to spit out a new state and signal. We create them by building a combinational circuit (so kmaps) and adding sequential logic components, such as registers.

The combinational circuit generates a new state that gets stored in the sequential component, which is then fed back into the combinational circuit.

3.1. The Synchronous Finite State Machine

In the soda machine example for state machines, we could wait very long before having to put in the last coin into the machine and it would still work. However, most computers are different in that they are **synchronous**, meaning that the state transitions take place at identical fixed units of time.

This is often done with the use of the **clock**, which produces a value which alternatives between 0 and some voltage in a repeated sequence of identical intervals known as the *clock cycle*.

4. LC-3

4.1. Von Neuman Model

A type of computer architecture known as the **Von Neuman Model**, which stores both data and program instructions in the same memory unit.

Main Components:
1. I/O
2. Memory Unit
3. Control Unit
4. Arithmetic/Logic Unit

4.2. The PC

The **program counter**, or PC, stores the *address* or the next instruction, often a 16 bit value because the addresses are 16 bits.

Main ways that PC can gte updated:
- PC + 1
- Bus: PC set to value on bus
- **PC + Offset**: offset value is added to incremented PC value for branching

4.3. Macrostates

Each LC-3 operation goes through a cycle of three macrostate: Fetch-Decode-Execute

4.3.1. Fetch

Definition 4.3.1.1

Fetch is responsible for retrieving an instruction from memory and preparing it for easy access. It takes three different clock cycles
1. Take the address of the next instruction from PC and put it in MAR so we can access memory. Increment PC so it is ready for the next macrostate cycle.
2. Access the value of the instruction from memory and put it in the MDR, preparing it for the bus
3. Take the value from the bus and put it into the IR so it stores the value of the current instruction

4.3.2. Decode

Once the instruction is in the IR, the finite state machine (FSM) uses the instruction's opcode and typical FSM logic to decide which instruction to execute.

4.3.3. Execute

After the FSM decodes the instruction, it asserts signales through the data path to execute the specified operations.

4.3.4. Microstates

While each operation has the same fetch and decode cycle, its execute phase has differing amounts of microstates depending on the context.

4.4. LC-3 Instructions

Each instruction can be split into multiple parts. The first four bits are the opcode, which defines which operation we are doing.

There are three types of instructions:
- **Operate instructions** act on data, such as ADD and NOT
- **Data movement instructions** move data around inside the data path and to/from I/O devices
- **Control instructions** change the order in which the program will execute later instructions, such as JMP

4.4.1. Addressing Mode

Instructions are comprised of their **opcode** and **operands**. Operands are things such as our source/destination register, or offesets, and how these operands are used to complete an instruction determine their addressing modes.

4.4.1.1. Immediate bits vs register

When doing operate instructions specifically ADD and AND, the arithmetic operation can specify if the second operand in the addition will be either a 2nd source register or a sign extend immediate 5-bit value.

1. DR $\longleftarrow$ SR1 + SR2
2. DR $\longleftarrow$ SR1 + sext[imm5]

Note that imm5 is a sign extended 2′s complement, so it ranges from -16 to 15.

Whether the operate instruction uses register addressing or this immediate value is determined by bit 5 of the instruction. Bit 5 is sent to the SR2MUX, therefore choosing between this second source register immediate value.
`ADD = [0001] [DR] [SR1] [0] [00] [SR2]` vs

`ADD = [0001] [DR] [SR1] [1] [imm5]`

4.4.1.2. PC-relative vs Base + Offset vs Indirect

When we do these data movement instructions, we often take a trip to memory. However, how can we give an address (16 bits) when we only have 16 bits for the whole instruction?

Theorem 4.4.1.2.1

PC Relative

Pc relative calculates our memory address from `mem[PC* + PCOffset9]`

Since our given offset is only nine bits, we can only load addresses $[-256, 255]$ from our current PC

`PC*` denotes that our PC has been incremented. This is already done by default in fetch.

Theorem 4.4.1.2.2

Indirect

Indirect addressing calculates our memory address from `mem[mem[PC* + PCOffset9]]`

This means our final address is the value we found at `mem[PC* + PCOffset9]]`

Example 4.4.1.2.1

For example, if our PC = x3002 and our PCOffset9 = x00001, we go to mem[x3003]. If mem[x3003] has hex value x5000, we go to mem[x5000] and load/store the value to/from memory.

This allows us to theoretically get to any point in memory.

Theorem 4.4.1.2.3

Base + Offset

Base + Offset addressing calculates our memory address from `mem[BaseReg + offset6]`

Example 4.4.1.2.2

For example, if R0 = x4000 (assuming its our base register) and our offset6 = x0005, we go to mem[x4005]

4.4.2. Control Instructions

Control instructions are used when we want loops or conditional statements, go-tos, etc. Basically, anytime we don't want to run our program line by line.

Theorem 4.4.2.1

BR - this branch instruction uses the CC to determine whether or not to branch to a new PC value. For example based on a CC register value we might update our PC value, otherwise we use our current `PC*` as our PC.

Theorem 4.4.2.2

JMP - this *always* updates the PC value, by doing PC $\longleftarrow$ BaseReg.

Fact check?

4.4.3. List of Some Instructions

Theorem 4.4.3.1

LD: Opcode 0010

LD is the load instruction. It loads the value at the memory address PC + PCOffset9 and stores it in a destination register.

`DR = mem[PC* + PCPOffset9]`

Also sets CC

Theorem 4.4.3.2

LDI: Opcode 1010

`DR = mem[mem[PC* + PFOffset9]]`

Allows you to access memory further from the PC

Also sets CC

Theorem 4.4.3.3

LDR: Opcode 0110

Adds the value of base register and a 6-bit 2's complement number to determine address in memory to load into destination register.

`DR = mem[BaseR + Offset6]`

Also sets CC

Theorem 4.4.3.4

LEA Used to calculate the effective address of an operand in memory and load that address into register without actually accessing the memory.

DR ⟵ PC + PCOffset9

Theorem 4.4.3.5

ST Puts the contents of a register into memory at the specific address (PC + offset)

4.5. Datapath

Definition 4.5.1

The **bus** is a 16-bit wire on the datapath that is connected to most of the major components in the LC3, and is used as a highway to transmit data. We can only have one signal on the bus at a time, which we control with **tri-state buffers**.

Definition 4.5.2

The **control code register** is a special register that holds 3 bits (N, Z, P) that tells us if an expression evaluates to negative, zero, or positive, using 1-hot representation.

Note 4.5.1

The following instructions set the condition code: ADD, AND, NOT, LD, LDR, LDI

Definition 4.5.3

The **finite state machine** decides the value of the next state based on the current state and current input (action).