**FIT1047 PASS WEEK 4**

Boolean Logic
Logic gates
Logisim
MARIE (basic)

## Boolean Logic, Logic gates, Logisim

- Give a boolean expression
  - Reduce using boolean identities
  - Reduce using K-maps
  - Construct using Logisim

| Identity Name | AND Form | OR Form |
|---|---|---|
| Identity Law | $1x = x$ | $0+x = x$ |
| Null (or Dominance) Law | $0x = 0$ | $1+x = 1$ |
| Idempotent Law | $xx = x$ | $x+x = x$ |
| Inverse Law | $x\bar{x} = 0$ | $x+\bar{x} = 1$ |
| Commutative Law | $xy = yx$ | $x+y = y+x$ |
| Associative Law | $(xy)z = x(yz)$ | $(x+y)+z = x+(y+z)$ |
| Distributive Law | $x+yz = (x+y)(x+z)$ | $x(y+z) = xy+xz$ |
| Absorption Law | $x(x+y) = x$ | $x+xy = x$ |
| DeMorgan's Law | $\overline{(xy)} = \bar{x}+\bar{y}$ | $\overline{(x+y)} = \bar{x}\bar{y}$ |
| Double Complement Law | $\bar{\bar{x}} = x$ | |

Using Boolean identities, simplify C + (BC)'. Does it give True or False?

| Expression | Rule used |
|---|---|
| C + (BC)' → C + (B' + C') | DeMorgan's (AND)<br>(AB)' = A' + B' |
| C + (B' + C') → C + B' + C' | Associative (OR)<br>(A + B) + C = A + (B + C) |
| C + B' + C' → C + C' + B' | Commutative (OR)<br>A + B = B + A |
| C + C' + B' → 1 + B' | Complement (OR)<br>A + A' = 1 |
| 1 + B' → 1 (True) | Null (OR)<br>1 + A = 1 |

# FIT1047 PASS WEEK 4

Boolean Logic
Logic gates
Logisim
MARIE (basic)

Using boolean identities, simplify X = ABC + A'B + ABC'.

| Expression | Rule used |
|---|---|
| ABC + A'B + ABC' → ABC + ABC' + A'B | Commutative (OR)<br>A + B = B + A |
| ABC + ABC' + A'B → AB(C + C') + A'B | Distributive (OR)<br>A(B + C) = AB + AC |
| AB(C + C') + A'B → AB + A'B | Complement (OR)<br>A + A' = 1 |
| AB + A'B → (A + A')B | Distributive (OR)<br>A(B + C) = AB + AC |
| (A + A')B → 1B | Complement (OR)<br>A + A' = 1 |
| 1B = B | Identity (AND)<br>1A = A |

**Using K-maps:**

- No group contains 0
- Groups cannot be diagonal
- Grouping in $2^k$ (1, 2, 4, 8...)
- Each group should be as large as possible
- Groups can overlap
- Each "1" must be part of a group
- Groups can warp around map (left & right)
- There should be as few groups as possible

Simplify F(A,B,C) = ABC + A'BC + ABC' using K-maps:

| | | BC | | | |
|---|---|---|---|---|---|
| | | 00 | 01 | 11 | 10 |
| A | 0 | 0 | 0 | 1 | 0 |
| | 1 | 0 | 0 | 1 | 1 |

**X = BC + AB**

**FIT1047 PASS WEEK 4**

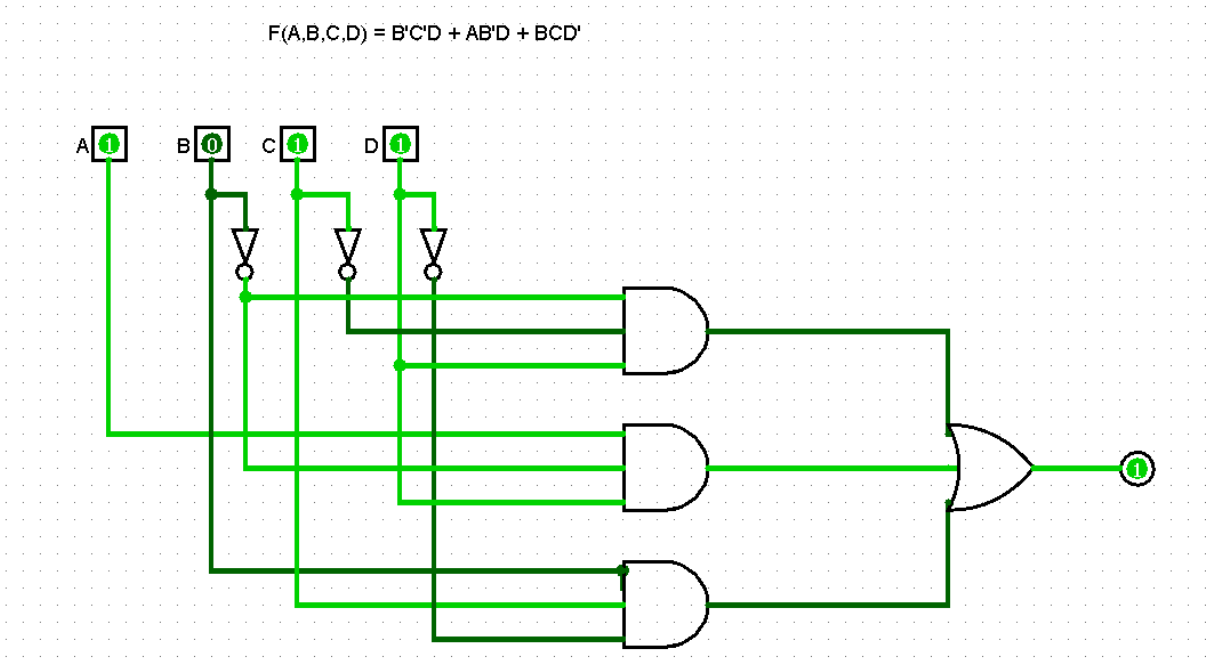
Boolean Logic
Logic gates
Logisim
MARIE (basic)

Simplify function F(A,B,C,D) = A'B'C'D + AB'C'D + AB'CD + A'BCD' + ABCD' using SOP' , then use Logisim to build the simplified circuit.

| | | CD | | | |
|---|---|---|---|---|---|
| | | 00 | 01 | 11 | 10 |
| AB | 00 | 0 | 1 | 0 | 0 |
| | 01 | 0 | 0 | 0 | 1 |
| | 11 | 0 | 0 | 0 | 1 |
| | 10 | 0 | 1 | 1 | 0 |

**F(A,B,C,D) = B'C'D + AB'D + BCD'**

Logisim circuit:



**FIT1047 PASS WEEK 4**

Boolean Logic
Logic gates
Logisim
MARIE (basic)

**MARIE Instructions**

| Type | Instruction | Summary |
|---|---|---|
| Arithmetic | **Add X** | Adds value in AC at address X into AC,<br>AC ← AC + X |
| | **Subt X** | Subtracts value in AC at address X into AC,<br>AC ← AC - X |
| | **AddI X** | Add Indirect: Use the value at X as the actual address of the<br>data operand to add to AC |
| | **Clear** | AC ← 0 |
| Data Transfer | **Load X** | Loads Contents of Address X into AC |
| | **Store X** | Stores Contents of AC into Address X |
| I/O | **Input** | Request user to input a value |
| | **Output** | Prints value from AC |
| Branch | **Jump X** | Jumps to Address X |
| | **Skipcond (C)** | Skips the next instruction based on C: if (C) =<br><br>- 000: Skips if AC < 0<br>- 400: Skips if AC = 0<br>- 800: Skips if AC > 0 |
| Subroutine | **JnS X** | Jumps and Store: Stores value of PC at address X then increments PC to X+1 |
| | **JumpI X** | Uses the value at X as the address to jump to |

**FIT1047 PASS WEEK 4**

Boolean Logic
Logic gates
Logisim
MARIE (basic)

| | | |
|---|---|---|
| Indirect Addressing | **StoreI** | Stores value in AC at the indirect address.<br>e.g. StoreI addresspointer<br>Gets value from addresspointer, stores the AC value into the address |
| | **LoadI** | Loads value from indirect address into AC<br>e.g. LoadI addresspointer<br>Gets address value from addresspointer, loads value at the address into AC |
| | **Halt** | End the program |

1. **What does the following MARIE program do?**

```
        Load X
        Add Y
        Output
        Halt

X,      Dec 2
Y,      Dec 3
```

- It performs the addition between X (value 2) and Y (value 3) then outputs the result.

2. **Write a program that subtracts value Y from X, then store it in a new variable and output the variable.**

```
        Load X
        Subt Y
        Store Z
        Load Z
        Output
        Halt

X,      Dec 8
Y,      Dec 2
Z,      Dec 0
```

3. **Write a program that accepts two 2 number inputs then output the larger number.**

Pseudocode:

```
x = input()
y = input()
if x > y:
      print(x)
else:
      print(y)
```

MARIE code:

```
      //x = input()
      Input
      Store X
      //y = input()
      Input
      Store Y

      //if x > y:   ## x - y > 0
      Load X
      Subt Y
      Skipcond 800 // skip next line if > 0 == if x > y: goto next
      line
      Jump PrintY
      Jump PrintX

      PrintX, Load X
      Output
      Halt

      PrintY, Load Y
      Output
      Halt

      X, Dec 0
      Y, Dec 0
```

**FIT1047 PASS WEEK 4**

Boolean Logic
Logic gates
Logisim
MARIE (basic)

Extra!
Simplify F(A,B,C) = A'B + BC' + BC + AB'C'.

$F = A'B + BC' + BC + AB'C'$

x = x+x (idempotent)

$A'B + (BC' + BC') + BC + AB'C'$

$A'B + (BC' + BC) + BC' + AB'C'$

$A'B + B(C + C') + C'(B + AB')$

$A'B + B + C'(B + A)$

$B(A' + 1) + C'(B + A)$

a' + 1 = 1 (null)

$B + BC' + AC'$

$B(1 + C') + AC'$

$B + AC'$

$B + AB'$

$B + B'A$

$B(1 + A) + B'A$

$B + BA + B'A$

$B + A(B + B')$

$B + A$

## WEEK 5 :

**Control unit** : coordinate components

**ALU**: perform arithmetic calculation

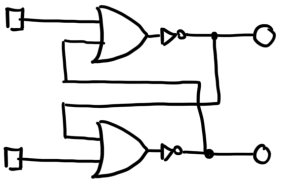
**Distinctive feature** :

Perform instruction and stored in the same memory space

If selector is 0, output will depend on the data received

**Set / Reset Latch** :

feedback loop, output is your input



**memory formula**: $\log_2[n]$

$2K \times 8 = 2 \times 2^{10}$ location $\times$ 8 bits per location

$= 2^1 \times 2^{10} \times 2^3$

$= 2^{14}$

**D - flip flop**

0 = read
1 = write

↳ sequencial circuit

↳ store one bit of information

↳ can be read and changed on a later point of time

4 address needs 2 bit : $\log_2(4) = \log_2(2^2)$

$= 2\log_2(2)$

$= 2$

**MARIE : 1 word = 16 bits**

**How many bits are in 32 Gibit ?**

1 Gibit : $2^{30}$ bits

$32 \times 2^{30} = 2^5 \times 2^{30}$

$= 2^{35}$ bits #

**I/O Control Methods** - When to do I/O?

| | |
|---|---|
| **Programmed I/O** (software responsible) | • **Checks for I/O new data periodically**<br>• Simple (no extra hardware)<br>• Full control polling - prioritise certain I/Os<br>• CPU constantly busy (to check for I/O) |
| **Interrupt-based I/O** (hardware responsible) | • **Hardware notifies CPU when new I/O data available**<br>• CPU interrupts program and jumps to special subroutine to process I/O request, then continues as normal.<br>• Programmers don't need to be aware of I/O. |
| **Direct Memory Access (DMA) I/O** | • **CPU delegates memory transfer operations to a dedicated *controller***<br>• Example: hard disk controller copies file directly from disk to RAM; graphic cards fetch image directly from RAM → CPU free to do other work!<br>• CPU and DMA share the same data bus - only 1 performs memory transfer at a time. |

**memory-mapped :**

- memory & I/O are treated the exact same way
  - ↳ same space, same address

**instruction-based :**

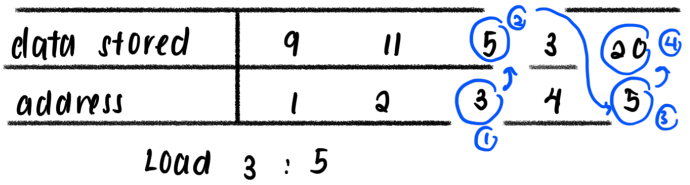
- memory & I/O are not treated as the same thing
  - ↳ NOT same space, same address
  - ↳ need different instruction to access

**MARIE - Indirect Addressing**

| | |
|---|---|
| **LoadI X** | Loads value stored at address of address X into AC |
| **JnS X** | Stores PC at address X and jumps to X+1 |
| **JumpI X** | Uses value at X as the address to jump to |

**MARIE - Subroutine**

Simple example: Print value X using subroutine PrintX.




Load 3 : 5

Load I3 : 20 → HOW? ① load 3, get 5 which is stored as target address ② ③ load target address 5 which is where 20 is stored.