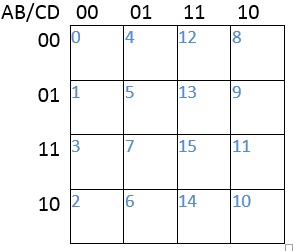
# Today:

* Course changes due to Covid and other announcements
* More Quine-McCluskey
* Start on a computer

## Course changes due to Covid and other announcements

* Labs will be remote starting Wednesday
* Exam should be back tomorrow by noon

## More Quine-McCluskey

Let’s do F(A,B,C,D)=Σ(2,4,10,11,12,13,15)

Steps are:

1. Generate Prime Implicants
2. Construct Prime Implicant Table
3. Reduce Prime Implicant Table (iterate until done…)
   1. Remove Essential Prime Implicants
   2. Row Dominance
   3. Column Dominance
4. Solve Prime Implicant Table

## Step 1: Generate Prime Implicants

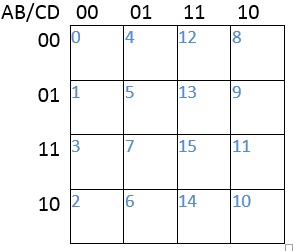
List minterms by their “Hamming weight”—that is the # of 1s in them. Then do a pair-wise check to see which terms can be combined with the grouping in front of them. Put a check next to each minterm that can be combined with another minterm.

Construct a prime implicant table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |

Would do further steps, but…

Now let’s say we have this one: Let’s do F(A,B,C,D)=Σ(0,4,5,9,10,11,13). Create the Prime implicant table using the Kmap.



1 1

1 1 1

1

1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |

1. Reduce Prime Implicant Table (iterate until done…)
   1. Remove Essential Prime Implicants
   2. Row Dominance
   3. Column Dominance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Start on a computer

Instruction set:

|  |  |  |
| --- | --- | --- |
| **Instruction name** | **Opcode** | **Effect** |
| halt | 0 | PC = PC+4 stop executing instructions |
| add | 1 | PC = PC+4 memory[addr0] = memory[addr1] + memory[addr2] |
| sub | 2 | PC = PC+4 memory[addr0] = memory[addr1] - memory[addr2] |
| mult | 3 | PC = PC+4 memory[addr0] = memory[addr1] \* memory[addr2] |
| div | 4 | PC = PC+4 memory[addr0] = memory[addr1] / memory[addr2] |
| cp | 5 | PC = PC+4 memory[addr0] = memory[addr1] |
| and | 6 | PC = PC+4 memory[addr0] = memory[addr1] & memory[addr2] |
| or | 7 | PC = PC+4 memory[addr0] = memory[addr1] | memory[addr2] |
| not | 8 | PC = PC+4 memory[addr0] = ~memory[addr1] |
| be | 9 | if (memory[addr1] == memory[addr2]) {  PC = addr0  } else {  PC = PC+4  } |
| bne | 10 | if (memory[addr1] != memory[addr2]) {  PC = addr0  } else {  PC = PC+4  } |
| blt | 11 | if (memory[addr1] < memory[addr2]) {  PC = addr0  } else {  PC = PC+4  }  Comparisons take into account the sign of the number. E.g., 16'hffff (-1) is less than 16'h0000 (0). |

Instructions are spread out over 4 addresses. Opcode, addr0, addr1, addr2.

|  |  |
| --- | --- |
| Location | Data |
| 0 | 1 |
| 1 | 10 |
| 2 | 20 |
| 3 | 30 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |

For example if memory holds the data on the right, then what is the instruction supposed to do?

Write a short program that performs A=B+C+D where A is location 50, B is location 51, C is location 52 and D is location 52.

|  |  |
| --- | --- |
| Location | Data |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |

Let’s look at a datapath that can do these instructions. We’ll only look at add in class, but it gives the idea.



**Maddr**

Control points:

PC\_drive, plus1\_drive, ALU\_drive, addr0\_drive, addr1\_drive, addr2\_drive, memory\_drive.

PC\_en, op1\_en, op2\_en, opcode\_en, addr0\_en, addr1\_en, addr2\_en, Maddr\_en, memwrite\_en

Draw a state machine for add. What part would be in common with the other instructions?