# Industry Selected

Agriculture industry-Apple farm.

# Company Background

New Zealand Apple Farm

The apple production in New Zealand is a huge sector for such a small country. The estimated crop for 2014 was close to 500 000 tones, which represents around 1 400 000 bins. If you consider that an average picker will pick about 200 bins over his season, it represents at least **7000 apple picking jobs**.

The New Zealand apple industry is seriously **focused on the export market** (about 60% of the production). It means that the quality control during the harvest is often stricter than in the countries which are more focused on the local market. There are about **500 apple growers in New Zealand**.

# Functions

## User Login Function

* 1. This function will prompt for user login to begin using the system. Verify username and password.

## Display Employees/Pickers Details

* 1. This function will display one picker in details such as hour of picking, number of bins they picked, and their total salary.

## Apple picker’s salary calculator

* 1. Apple picking is paid on contract, meaning that instead of being paid a fixed hourly rate, picker will get paid depending on their production (i.e. how many bins do they picked).
  2. Hourly rate = hour worked \* (fixed rate per hour)
  3. Commission = [Price of apples] (number of bins of apple \* price of apples per bin) \* 8%
  4. Picker’s Salary = hourly rate \* commission

## Apple price calculator

* 1. This function will calculate the total price of the apple collected and show the number of apples to be export and the profits.
  2. Price of apples = price of apple per bin \* number of bins

## Types of apples

|  |  |
| --- | --- |
| Apple Type | Apple Price |
| 1. Braeburn | 172.12 |
| 1. Fuji | 228.23 |
| 1. Granny Smith | 144.45 |
| 1. Honeycrisp | 128.56 |
| 1. NZ Beauty | 112.67 |
| 1. NZ Queen | 114.78 |
| 1. Red Delicious | 124.89 |
| 1. Royal Gala | 108.90 |

## Sample reports

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Employee’s Weekly Hours Report | | | |
|  | Hours | Hour Rate | Total Hour Pay (Hour \* Hour Rate) |
| Monday | 8 | 7 | 56.00 |
| Tuesday | 8 | 7 | 56.00 |
| Wednesday | 6 | 7 | 42.00 |
| Thursday | 0 | 7 | 00.00 |
| Friday | 9 | 7 | 63.00 |
| Saturday | 9 | 7 | 63.00 |
| Sunday | 0 | 7 | 00.00 |
| Total | 40 |  | 280.00 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2. Employee’s Weekly Task | | | | |
|  | Apple Type | Bin | Bin Rate | Bin Value |
| Monday | Honeycrisp | 3 | 128.56 | 385.68 |
| Tuesday | NZ Queen | 4 | 114.78 | 459.12 |
| Wednesday | Royal Gala | 2 | 108.90 | 217.80 |
| Thursday |  | - | ---.-- | ---.-- |
| Friday | Fuji | 5 | 228.23 | 1141.15 |
| Saturday | Granny Smith | 5 | 144.45 | 722.25 |
| Sunday |  | - | ---.-- | ---.-- |
| Total |  | 19 |  | 2926.00 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C. Employee’s Weekly Salary Report | | | | |
| Hour Pay | Bin Value | Commission Rate | Commission | Salary |
| 280.00 | 2926.00 | 8% | 234.00 | 514.08 |

# Simulated 32-bit division & multiplication on a 16-bit System

The one of the main objectives of this assignment is to understand how arithmetic operations works in assembly. Modern programming languages could calculate large numbers or complicated formulas, all within one line and comprehensible format. But in 8086 Assembly, we’re limited up to 16-bit multiplication and division.

For example, we want to calculate [FFFF16 \* A16 = 9 FFF616], the ‘9’ would get stored in DX and the ‘FFF6’ would get stored in AX. Yes, 8086 can handle that, since both multiplier and multiplicand are 16-bit, and the 32-bit resultant is then separated into two 16-bit registers.

## Problem statement

What if we wanted to “1 000016 \* A16”? We’ve looked up online as we knew the multiplicand doesn’t fit into AX and found “EAX \* 32-bit Source = EDX: EAX”. But 8086 is a 16-bit microprocessor, we don’t have EAX, EBX, ECX or EDX. At first, we almost wanted to give up and amend the proposal to adapt to 8086’s capability. As programmers, our job is to solve problems.

|  |  |  |
| --- | --- | --- |
|  | 000F | FFFF |
| x |  | A |
|  | 0009 | FFF6 |
| ~~0000~~ | 0096 |  |
|  | 009F | FFF6 |

And thus, began the idea of simulating 32-bit operations with 16-bit processes. Let’s start with:

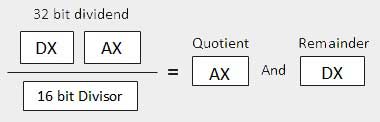
## Multiplication



We first, take the lower part of the multiplicand, and multiply it. From the example we can see ‘000916’ will be in DX and ‘FFF616’ will be in AX. We will save the DX in for final calculation. And then multiply the higher part of the multiplicand and multiply it. Add it with the previous value of DX and we’ll have our result.

## Division

|  |  |  |
| --- | --- | --- |
|  | 0019 | 9999 |
| A | 00FF | FFFF |
|  | FA |  |
|  | 0005 | FFFF |
|  | 0005 | FFFA |
|  |  | 0005 |



In 8086, we are able to have a 32-bit dividend (DX:AX) with a 16-bit divisor. But the result will be 16-bit as the quotient is in AX and remainder is in DX. If the quotient is more than ‘FFFF16’ then everything will go wrong. So again, we divide the higher part of the dividend, and move our quotient to the higher result. We take the remainder and move it to DX, and the lower dividend to AX. This way, the DX is always smaller than the divisor thus the result will always be 16-bit. Divide and take to quotient to the lower result. And thus, we have our 32-bit result.

# Pointers

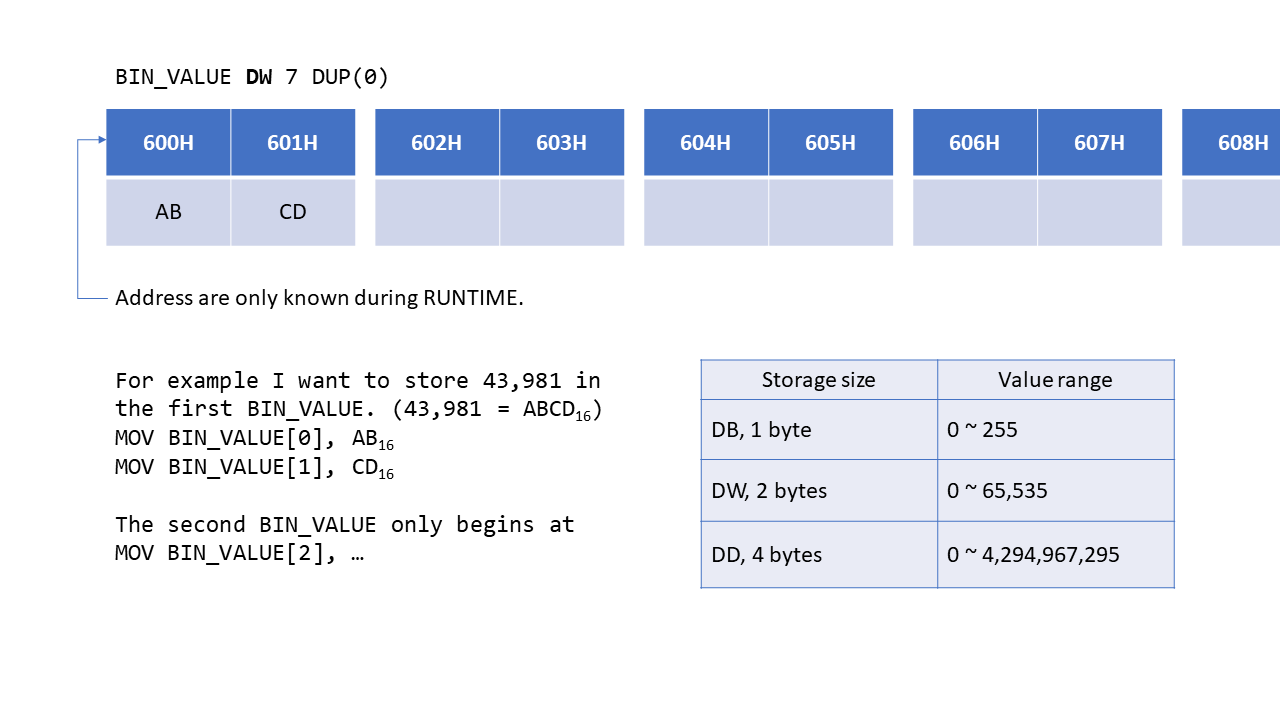
In our second semester AACS1084 Programming Concept and Design II, we were quite familiar with C. We remembered Chapter 4.9.1, Pointers. At first, WEwas not quite understood what it was for other than storing user input.

But it’s different when we go to Assembly, we even dove deeper into addressing, Segment: Offset, Direct, Indirect. And we would use them when we want to access arrays. Arrays in C, when we want to get the next value we would just add 1 to the index, regardless of the variable size.

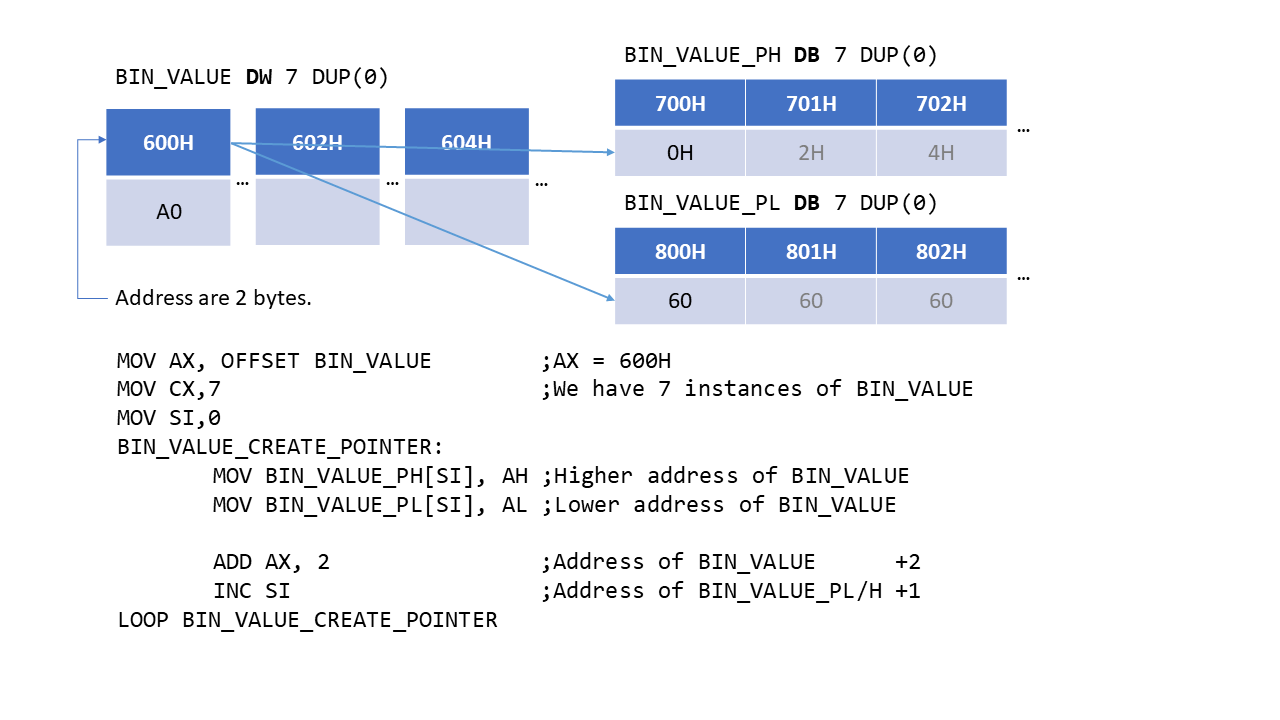
## Problem statement

Now here’s the problem, array is different in Assembly. Increment of the index depends on the variable size. Array of 10 bytes would require index increment by 1. Array of 10 words would require index increment by 2. For example, we have to print out 7 days, we have an array of string, which stores Monday until Sunday. Each string has equal length of 10 characters, we would have to have an index increment by 10 as each character allocates one byte.

Then we have to print an array of strings, which contains apple names, right next to the days. Each apple names are 15 character in length. It was quite easy as we just have another index which increment by 15 each loop. Two indexes, two arrays, problem solved. Or so we thought.



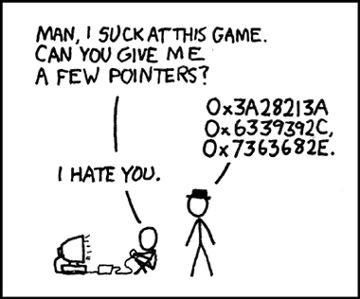
Next is to print out an array of **words**, at this rate we knew we are going to either run out of index to play with or its going to be hundreds of lines to change index values from memory whilst updating each and every single index. Spaghetti code, is what we wanted to prevent. And thus, we reached into a conclusion, we take the addresses we need and store them.



Since address are 2 bytes in size, we would then store the higher address into one array, then lower address into another. Both are array of bytes, which increment by one. So, when we refer to the 4th word in an array, We would just have to take the 4th higher address and the 4th lower address and then we would find the 4th word in an array. Sounds hideous but this alone could save a lot of code in a loop, which is very critical as loop has a limited number of bytes it could jump.

So, in our program, right after the user entered the correct password, it would call a procedure which would initialize pointers of words or double words.

With that being said, here’s a joke about pointers we’ve found on r/ProgrammerHumor.



# 7 Digit Input

During development of the program, a popular question was asked around our course. “How to use string to calculate ah?”. We’ve found out that most people would just use buffered input (0AH) to get user input values. Have a user input like “4321” in string form they would just need to [(4\*1000) + (3\*100) + (2\*10) + 1].

## Problem statement

Buffered input (0AH) will get any user input, and store it as a string. But what if, the user input a character? Or the user inputted more value than it could handle? Would the first digit of “321” or “4321” always get multiplied by 1000?

So, we didn’t use buffered input (0AH) to get user input in any part of our program, even password input because we wanted to hide user input with an asterisk (\*). We use direct char read but without echo (07H). Reason number one, we could limit user input to only digits, if UserInput < 0 or 9 < UserInput equals true, then we could just ignore the Input. Second reason is, we will have the UserInput as a integer, no need for convert string to int (parsing). Third reason is, we want to display UserInput in our format which have two decimal points.

Of course, this method doesn’t come without challenges, first our input value will be initialized with 0. Then, we’ll print out a string which doesn’t have to usual new line characters appended before the string. But it has 0DH at the beginning of the string, 0DH stands for carriage return, returning to the beginning of the current line without advancing downward. This string is in a loop, so line will never get advanced downward without exiting the loop. Then we’ll print out the input value which is 000(.)00.

## Input

When user input ‘5’, the current value will get multiplied by 10 (0 \* 10 = 0). And then added with the input value (0 + 5 = 5), and loop again. 0DH will make the cursor to the leftmost of the screen, and overwrites the current line with the new input value, which will display 000(.)05.

So, if the next UserInput = 9, (5 \* 10) + 9 = 59.

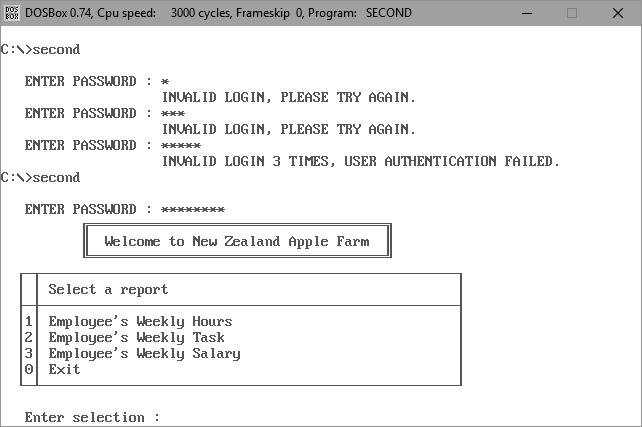
## Backspace

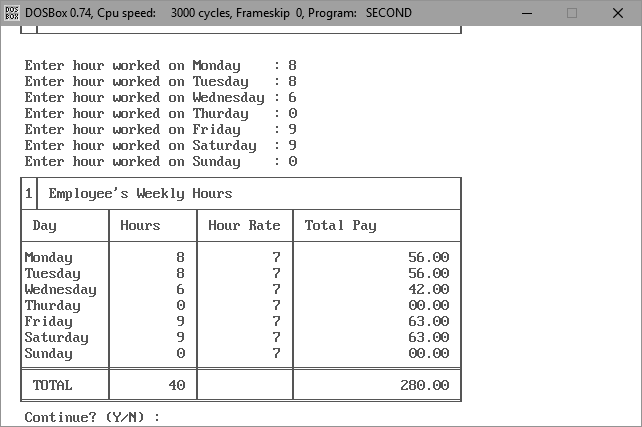
When user input backspace key (8), we will take the current value, and divide by ten, discard the remainder as the user demands. And display the new value.

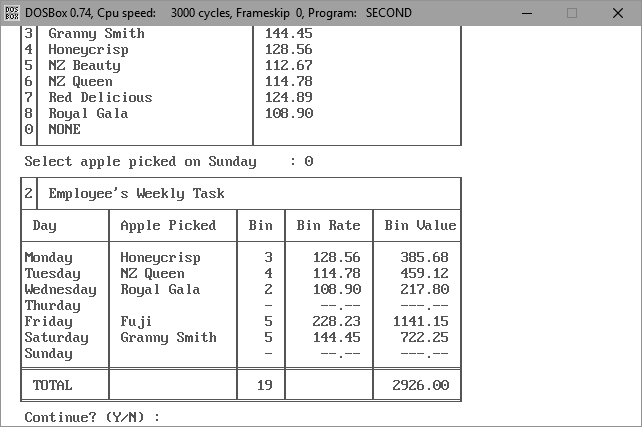
Due to our display limitation, the max value the program can display is 32767(.)00. Which is quite abnormal for our program as no picker would be able to pick out that many bins of apples in one week. So, when the current value is more than 3276(.)70, and the user inputted another value, the program would just give the user the largest value it can hold which is 32767(.)00.

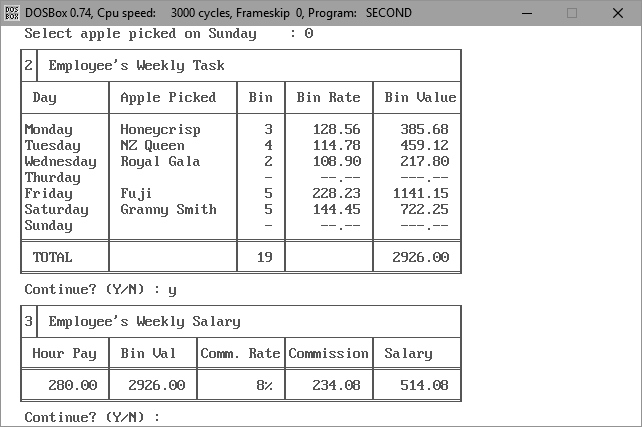
Both multiplication and division are using simulated 32-bit processing. Without it, our max input will probably be 655(.)35.

# I/O Design

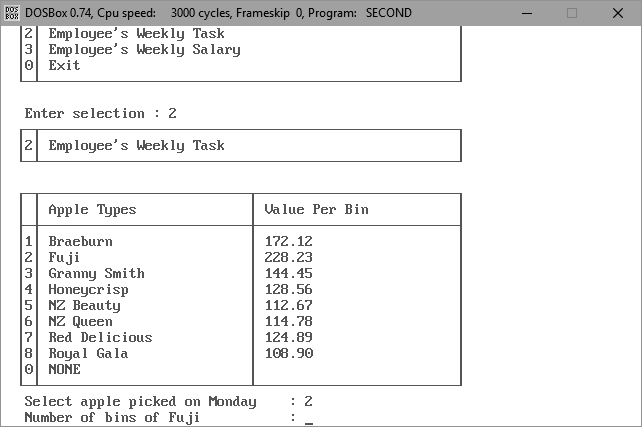
Password and Menu

Employee's Weekly Hours

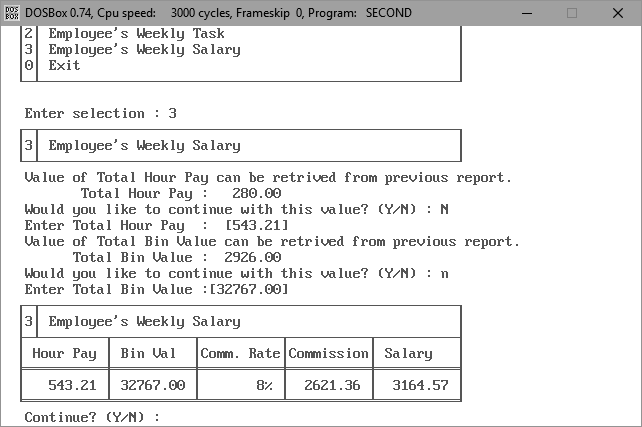
Employee's Weekly Task

Employee's Weekly Salary 

## User Input Apple Types and number of Bins



## User Input Multi Digit



# User Guide

User get DOSBOX installer here <https://www.dosbox.com/download.php?main=1> and install DOSBOX.

Install the 8086 folder inside user’s “C:/” directory.

Upon running DOSBOX, enter “mount C: C:/8086”, it should prompt “Drive C is mounted as local directory C:/8086/”

Enter this “C:”, and user should notice the “Z:\>” on the left changes to “C:\>”

The password is “password”, it’s literally telling the user to enter ‘’password”.

From there user should be able to see the menu of report it can generate.

1. Employee's Weekly Hours
2. Employee's Weekly Task
3. Employee's Weekly Salary
4. Exit

## Employee's Weekly Hours Report

User are prompted to enter hours worked by the employee day by day. User are able to enter 0 if the employee is absent or weather is bad on that day.

Once 7 days are inputted, it should generate a report

# Reference

1. [en.wikipedia.org/wiki/Code\_page\_437](https://en.wikipedia.org/wiki/Code_page_437)

We use this page to view the list of available ASCII characters we could use in our program. It is very useful for us as we wanted to tabulate and display data like we planned in our proposal.

1. [www.tutorialspoint.com/assembly\_programming/assembly\_arithmetic\_instructions.htm](http://www.tutorialspoint.com/assembly_programming/assembly_arithmetic_instructions.htm)

This page simplifies how Arithmetic Operations in assembly works. Which enable us to understand how Assembly do multiplication and division. It was also this page that made us realize 8086 doesn’t have EAX, EBX. ECX, EDX. Initially, we wanted to give up calculating 32bit values. But then we tried to stimulate 32bit division/multiplication referring to this page (for quite some time) and we succeeded (to a certain degree).

1. [www.tutorialspoint.com/microprocessor/microprocessor\_8086\_instruction\_sets.htm](http://www.tutorialspoint.com/microprocessor/microprocessor_8086_instruction_sets.htm)

Every 8086 instructions can be found here. CALL enabled us to make short jumps whilst executing more than 127 bytes of code, also to ease the strain on our eyes. ADC enabled us to preform 32bit addition and adding the carry (if there’s any) at the same time.