#  3P92 Project

# 2020

Ver 3.0

The 3P92 project will consist of the design and implementation of an 8x8 micro processor. Software used for the implementation will be Logic Circuit, available from the web as a free download.

MPC Architecture

A

B

PSW = 0xC8

C

SR Vector

Special MCode addr. To handle ISR, controlled by h/w see note 2

I

SP

Micro Store

Outputs a series of control lines which effect

1. ALU function
2. Register i/o
3. Memory r/w

MPC

IR

PC

MBR

Memory

MAR

PSW (process status word) \_ \_ \_ I C V Z N (LSB is to the Right).

N is set when a computation within the ALU causes the result to be negative

Z is set when a computation within the ALU causes the result to be zero

V is set when a computation causes an arithmetic overflow.

C is set when an arithmetic computation results in a carry, using 2’s compliment arithmetic.

I is set when a hardware interrupt is detected.

**Assembly Level Language Instruction Conventions**

Each Operand can take the following format:

Operator Defines an inherent instruction.

Operator X Instruction will operate on the immediate Operand X.

Operator A Instruction will operate on the Direct Memory Address A.

Operator A, X Instruction has 2 Operand, Direct and Immediate.

Operator A,B Instruction has 2 Operand, both direct.

 X = Literal, defined as a signed or unsigned base 2 number.
 A, B are memory addresses.

In cases where there are 2 operands, the first maybe a source/destination and the second source.

The CPU functions as a memory to memory architecture, all operators will either specify a literal or a memory address.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mnemonic** | **Binary Code** | **Description** | **Effect on PSW****A B & C** | **Supplemental Info** |
|  |  |  |  |  |
| NOP |  | No operation |  | Cycle Waster. |
|  |  |  |  |  |
| HLT |  | Disables the clock, halting further computation. |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| ADD A,X |  | (A) = (A) + X | N=C7Z=Cx all zeroV=A7.B7.C7’+A7’.B7’.C7C = A7.B7+B7.C7’+C7’.A7 |  |
| ADD A,B |  | (A) = (A) + (B) | Same as above |  |
|  |  |  |  |  |
| SUB A,X |  | (A) = (A) - X | N=C7Z=Cx all zeroV=A7.B7’.C7’+A7’.B7.C7C = A7’.B7+B7.C7+C7.A7’ |  |
| SUB A,B |  | (A) = (A) – (B) | Same as above |  |
|  |  |  |  |  |
| INV A |  | (A) = (A)’ | N=C7Z=Cx all ZeroV=0C=1 | 1s complement |
|  |  |  |  |  |
| NEG A |  | (A) = (A)’ +1 | N=C7Z=Cx all ZeroV=C7.C6’.C5’.C4’.C3’.C2’.C1’.C0’C=any Cx=1 | 2s compliment |
|  |  |  |  |  |
| AND A,X |  | (A) = (A) and X | N = C7Z = Cx all ZeroV=0 | Bit wise |
| AND A,B |  | (A) = (A) and (B) | Same as above |  |
| OR A,X |  | (A) = (A) or X | Same as above |  |
| OR A,B |  | (A) = (A) or (B) | Same as above | Bit wise |
|  |  |  |  |  |
| CMP X |  | Compare X to 0 | N=C7Z=Cx all ZeroV=A7.B7’.C7’+A7’.B7.C7C=A7’.B7+B7.C7+C7.A7’ | Force PSW |
| CMP A |  | Compare (A) to 0 | Same as above |  |
| CMP A,X |  | X == A | Same as above |  |
| CMP A,B |  | A == B | Same as above |  |
| BR X |  | PC = PC + X |  | X is signed |
| BRZ X |  | if Z then PC=PC+X |  | X is signed |
| BRN X |  | if N then PC=PC+X |  | X is signed |
| JMP X |  | PC = X |  | X is unsigned |
|  |  |  |  |  |
|  |  |  |  |  |
| MOV A, X |  | (A) = X |  |  |
| MOV A,B |  | (A) = (B) |  |  |
|  |  |  |  |  |
| LSP X |  | SP = X |  |  |
| LSP A |  | SP = (A) |  |  |
| SSP A |  | (A) = SP |  |  |
|  |  |  |  |  |
| PSH X |  | (SP) = X | side effect after | SP -= 1 |
| PSH A |  | (SP) = (A) | side effect after | SP -= 1 |
| POP A |  | (A) = (SP) | side effect before | SP+= 1 |
|  |  |  |  |  |
|  |  |  |  |  |
| WTI  |  | Wait for Interrupt | I = h/w interrupt recieved | CPU stalls until I |
| JISR |  | Jump to ISR, Hard coded Micro instruction which will initiate the ISR. see A2 for details | Not an assembly level instruction. But must be micro coded. |  |
| JSR X |  | Push PSWPush PCPC = X |  | SP is modified accordingly |
| RTS |  | Restore registers.Return from sub-routine |  | SP is modified accordingly |
|  |  |  |  |  |
|  |  |  |  |  |

Note 1: Hardware interrupt, I is set, Current instruction finishes executing. When Fetch unit writes to MPC, the MPC receives a hardcoded MCode address which handles the jump to interrupt service routine.

Note 2: PSW represents the CPU’s status flags. In order to simplify the process, memory address 0xC8 shadows the PSW. Reads and Writes to 0xC8 (200 Dec) will be reflected in the PSW. This allows for bit masking and testing of individual flags to be carried out using existing assembly level instructions.

## Assignment 1: Oct 26th 4:00 p.m. EST.

This part of the assignment will focus on developing basic components of our CPU. Three components will make up the majority of the basic buildings blocks for our CPU.

a). An 8 bit register. This register would be identified as the IR, A, B, C, MAR, MBR & SP. In effect most of the same circuit will be used for all. Using Logic Circuit create an 8 bit register with the following properties, as it applies to the specific register.

* Register A , B, IR and MAR many be connected to the data bus when being written to, otherwise their output (A and B)(from Q) will be fed directly into the A and B pathway of the ALU. MAR will feed similarly into the memory unit. Using Logic Circuit design a register which has 8 data lines in - under the control of a tri-state device, when we write to the register, the tri-states are activated and the value of the data bus will be latched on the high cycle of the clock. Q will represent an output bus which will feed directly into the ALU once completed. See lecture for details
* SP and MBR are similar to A and B above except when reading (reading to Data bus), Q will appear on the Data bus. This register will have 2 control lines, 1 for reading, 1 for writing. As well as opposing tri-state for directional control.
* C is always connected to the ALU and will update according to the ALU function. It however, can be read from, thus requiring a tri-state to isolate it from the Data Bus.

b). Due to timing limitation of Logic Circuit, many of the standard tricks such as edge triggers and propagation delay do not work consistently to yield stable results. Thus, it is required to break the clock down into sub-cycles. Build your own clocking cct with the following properties:

C = defines a full clock cycle, high to low.

C1-4 define ¼ pulses of C.

See lecture for full detail.

In addition, your clock should contain an enable line (not shown), thus when enabled, the clock will generate pulses. This is required for the halt instruction. In Assignment 2 you will be asked to modify this cct again to improve functionality.

c). The PC is a very special case. It behaves as a normal register in most cases, with much the same properties as the SP and MBR. However, one of the most common operations performed on the PC is increment. This can be accomplished by running the contents of the PC through the ALU and adding 1, however a nifty h/w solution exists to increment the PC.

A PC with a built in counter is very useful. Take a look at the supplied circuit in the project directory, called Program Counter. It gives a firm framework to develop yours with the ability to increment its contents. Read the notations in the ccts for a full explanation of what is required and how it works.

The notes also describe what the PC should have in terms of i/o lines. 80% of the work is done, you get to do the other 20%.

In addition read the notes for the clock CLK unit. This was developed to overcome timing issues of the PC. It is believed that its functionality will be needed when we deal with other components which require strick timing sequences.

A tri-state buffer was also developed. Use it as a building block for your project.

d). Create an ALU which will perform the basic functions as outlined in the assembly level instructions. Use the example in class as a starting point. Consider creating a TT which will define for each bit the required output. The following operations should be implemented. A, B and C refer to the inputs and outputs of the ALU, not the registers A,B and C. The ALU functions should be supported by a decoder, 4 lines.

1. No Operation -- ALU will not perform any function. Acts as an ALU disable
2. Add with carry in and carry out. –affects flags V,C,N,Z
3. C = A -- data movement
4. C = B -- data movement
5. C = B’ -- basic not function, affects N,Z,V,C
6. C = A & B -- affects N,Z,V
7. C = A | B -- affects N,Z,V
8. C = 0 -- ALU, generates a 0, affects NZ
9. C = -1 -- ALU, generates a -1, affects NZ
10. C = 1 -- ALU, generates a 1, affects NZ
11. A+1 (INC x) -- C = A +1 i.e. C = A + (B =1),V,C,N,Z
12. A-1 (DEC x) -- C= A-1, C=A+(B=FF), ability to force B to 0xFF;affects N,Z,V,C
13. A-B -- C = A – B, C= A+(B’+1), affects N,Z,V,C
14. CMP A B -- Set N & Z as appropriate V,C
15. to 15 -- not used

The ALU will require 4 input lines which will be decoded to produce the functions as stated above. Functions 14-15 will not be used. For consistency within the class, assume that the decoder function will implement the above in the order given.

Much of the ALU function will be responsible for affecting the PSW. E.g. most operations except 2 & 3 affect the flags; these are data movement instructions through the ALU. Others are the result of a computation, e.g. ADD affects all and CMP some. This implies that there will need to be as much effort put forth to implement the Flags as the ALU function. Much of the ALU can be designed from the example ALU (see lecture slides), however, do not rely solely on this, since some extra functionality is required.

A basic block diagram of the ALU is as follows:

ALU

4 Lines from Micro store

A

B

C

PSW, NZVC

4 Lines from Micro store, same 4 that go to the ALU, determine which flags are enabled for modification.

For each ALU function the decoder will also activate logic to perform the bit logic for the PSW. In most cases this logic is a series of AND,OR and NOT functions of the bits of input A & B and output C. See the assembly level instructions for exact bit logic. Once the PSW flags are calculated they must be written to the PSW. It is important to note that not all calculations of the ALU affect all PSW flags. Thus, only select bits of the PSW will need to be active for writing, this is determined by the associated ALU function. For this assignment, your ALU must generate the appropriate NZVC, in that order to correspond to the PSW bits 0-3. In Assignment 2 you will put together the PSW ( I will help on that design). It will require the same 4 control lines that the ALU uses to enable correct writing. Basically, it will use the exact same decoder. Note: PSW flag calculation will be done in Assignment 2. For this assignment only worry about getting the ALU to correctly generate the C output based on input A & B and the function lines. This will be challenging enough.

Calculations on the ALU will happen on the positive cycle of the clock. At that moment the ALU logic should be set and the result written to the C latch. On the lower half of the clock the o/p latches will hold the result of the ALU. The same process will take place in the PSW. You can assume that operands written to A and B are immediately visible to the ALU, i.e. you do not have to enable A or B for reading, this makes the implementation of those registers trivial. C will be enabled for writing when the ALU receives a valid function (1-13).

### Proposed completion schedule

The creation and implementation of logic circuits can take some time. From the start of this assignment to the end you have 5 weeks of development. It is unreasonable to assume that you can complete all 4 parts in 2 days. In fact you can’t. The below schedule is a proposed set of completion dates for each part. If you are falling behind, you will need to pick up the pace.

a). Complete by Oct. 5 These dates will be updated

b). Complete by Oct. 12

c). Complete by Oct. 19

d). Complete by Oct. 26

Be sure to allocate 3-4 days to properly test. Part d) is very large and can consume a large quantity of time. Do not take this section for granted. Also worth noting, that future success in the project is governed by successful correct completion of each stage.

Submission:

Your submission will be a set of components which will be used for later use, i.e. A2. Each component which is built must have an appropriate test harness to show the marker, that your cct. does what it supposed to do. Each component must be in a package (chip with i/p and o/p defined and labelled). For each of the following components, have a test harness clearly labelled as follow, showing appropriate behaviour. Test harness will drive the chip and show correct function.

a) Label Test\_A, Should show that an input as supplied by a sensor is latched when write lines are enabled, on the high pulse of clock.

b) Label Test\_IR, Should show that an input as supplied by a sensor to the bus side for writing is latched by the IR. Should show that when read, the output of the register is written to a separate output. Remember, IR and MAR will behave similarly, with a unique input and output path. See block diagram.

c) Label Test\_MBR, Show that an input is latched when written, and that when read the output appears on the same bus lines. That is i/p and o/p will go down the same bus path. You may use the 8bit\_splitter cct. As supplied to aid in this.

d) Label Test\_C, This is the C latch from the ALU. Should via a sensor, a direct latch when written to, i.e. no need for tri\_state i/p. However, the o/p must be tri\_state to connect to the bus. Write will be controlled by ALU function, read by micro store.

e) Label Test\_Clock. Show your clock cct. With prescribed output. Testing should show an oscilloscope output with proper timing pulses.

f) Label Test\_PC. Show function of the PC as defined with inputs and outputs as well as the ability to increment when instructed to do so.

g) Label Test\_ALU. Show for each decoded ALU function the result C w.r.t. inputs A and B. Test harness should use sensor input which define A and B, and a probe defining C. Note, that sensors and probes can represent multiple lines. It may be sufficient to show defined inputs A and B, and then cycle through the ALU functions.

Package the above in a (one and only one) Logic Circuit project for the marker. Written instructions can be added if you feel this to be appropriate. Submit via Sakai.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ End of Assignment 1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Assignment 2: TBD

This part of the project will focus on extending and developing a working CPU.

a) In part 1 you developed an ALU, but were told that the PSW (setting of the flags) would be handled in part 2. Well here is part 2.

As part of A2 complete the logic of the ALU to support the flag calculations. Each ISA level instruction will leave the PSW in an altered state. Since each ISA instruction which uses the ALU will have one ALU operation, that operation will set the bits of the PSW in accordance to the ISA instruction. In the table which lists the ISA instructions is the logic which calculates the flag state. The ALU decoder (based on the ALU function selected) will also enable the correct bits of the PSW. Combinatorial logic will then produce the result based on the inputs A, B and output C of the ALU. This will be your first task. Ensure you create an appropriate test harness to ensure correctness.

The PSW is given. Read the documentation within the cct for detail.

b) A second issue which must be addressed is a user interface. This interface is fairly simple:

* Must be able to load Assembly level instructions into memory.
* Must be able to initialize the system (RESET), set a starting address.
* Start the system (START).

Logic Circuit does most of the loading and viewing, however, the issue of reset and run must be taken care of. When the power is applied <power on> in logic circuit, the system should be in an idle state. At this point, we should be able to hit the RESET button which will reset the PC and MPC to 0, as well as clearing any latches which are deemed necessary for the system to run. Once this is done we can run the cpu (START). The program will run until a HLT instruction is encountered, which will disable the clock.

To accomplish this we will have 2 buttons on our UI, START and RESET. Power on is handed by logic circuit.

Ensure that the clock circuit has the ability to enable and disable via a control line.

Ensure the PC can be reset to 0 as a starting address.

Wire the system together as shown on the above diagram. Use RAM as provided by Logic Circuit to implement a 128 byte memory at the low end of the memory map.

Concept: Put the assembly program into RAM using logic circuit. The RAM can be set to “Persist memory between runs” and thus allows one to enter hex values which will represent the ISA code. This also allows for the RAM image to be saved and then loaded from a file.

 Below, is a block diagram of the user interface:

RESET

START

Summary:

1. Modify the clock as described
2. Modify the PC as described
3. User Interface
4. Wire it up

c) The Micro Store is defined below: Currently up to date



The above is very similar to the MPU defined in the book. Obviously, there are differences, and thus different control lines are defined. Provided in the course website as MPC&MStore. It is tested, and comes with the usual disclaimer, test for yourself. The output MSo is a 32 bit output bus which represents the Micro Store as defined above. Only those control lines which are required to be external to the cct are defined. See the cct and notes. Connect this into your project. The MBR as mentioned in class can be read/written from 2 different buses. It is expected that at no time will it be connected to the data bus and to memory simultaneously. Thus, control line 21 and the decoder will allow the read and write connection to the data bus. This is similar to all registers. When dealing with memory, MBR should be controlled with lines 25 & 26. Thus an operation to write to memory can enable MBR to be read from, and visa versa. This will require a separate set the tri-state buffers to allow MBR to connect the memory.

The address lines from the MAR will address memory and the PSW.

No need to worry about hardware interrupts for now. This will be a quick addition once we get to that part.

d) To ensure the basics are working you will need to write some minimal code. We have a simple system, whereby execution will start with the PC=0 and MPC=0 as defined by a reset. The user program will start at 0 in memory. The first thing the MPC must do is fetch. Thus it makes sense the fetch routine starts at 0 in the micro store. It should look something like this.

MAR <- PC

rd.

IR<-MBR

PC+1

Load MPC from IR.

Subsequent Micro instructions will then execute with the following format:

Series of Micro Instructions

Fetch, defined by Next Address = 0x00

Write the Micro instructions to implement the following Assembly level code

MOV A, 2

ADD A, 3

HLT

e.g. Suppose A=0x0A, thus the memory contents of your program should look as follows:

MOV

0x0A

0x02

ADD

0x0A

0x03

HLT

...

To aid in the coding, you will find an excel spreadsheet in the resources section of the website. Use this to determine the MicroStore data. We will go over this in lecture. Obviously, you will need to implement a minimum number of micro instructions to allow the above assembly code to execute. Fetch, MOV, ADD and HLT.

If you enable the clock for a sufficient amount of time the program should stop executing at HLT. Hopefully, when you examine memory 0x0A it should contain 5.

Submission:

Submission is a work or don’t work scenario. Thus if you can show that 0x0A ends up with the value of 0x05 as a result of the calculation, then you can assume a very good mark. Here is a breakdown of the mark allocation.

Must create a test harness (named Part A) which shows that each major ALU functions correctly sets the PSW flags.

Create a user interface as described above which allows for the resetting and execution to be performed.

Create a circuit which will wire all major components together, as described.

Give your main project a descriptive name, and ensure the marker will know what the main circuit is. Write the micro code to implement the ISA instructions as described, execute the code. The marker will interrogate your RAM and ROM to see if it works and complies with the instructions given. Be sure that RAM starts in a state where 0x0A = 0x0.

Submit via Sakai

Remember: A Happy marker will make you Happy.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ End of Assignment 2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Assignment 3 (At time of presentation)

(Part A)

Continue to implement the ISA instructions as defined in the document. **A detailed description of the final stage will be given in lecture.**

 (Part B)

Add additional memory to your system so that the memory map will look as follows:

128 bytes

32
 bytes

0xE0

0xC0 = OutByte
0xC4 = InByte
0xC8 = PSW

We can use a 32 byte module for upper memory; this is where the interrupt service routine will be kept. The address 0xC0 and 0xC4 will be specialized registers for i/o, attached to the memory addresses as specified.

WTI – This command will effectively execute micro code which will put the CPU into an infinite loop until a h/w interrupt arrives. To break out of the loop, we require the ability to modify next address within the micro store. This system has already been set up for BRZ and BRN. The micro store has been modified so that when WTI is executed, it is expected that line 28 (N/Z’) is asserted. If I gets set, this signal will modify the 9th address bit allowing a jump within the MS. So WTI will have 2 entries into the MS. Low address entry will be an infinite loop with bit 28 set. The high address entry will clear 28, and specify a Fetch.

At this point I is set, and JISR micro code must be executed to force the ISR. This is accomplished (see block diagram) by the fetch. When I is set the next instruction will be pulled from a hard coded input. This can be accomplished by using the 8 bit Y connector (see MPU\_Resources) under the control of I (I=0 then pull from IR else hard coded JIRS instruction). The basic principle of JISR is as follows:

1. Disable interrupts, and clear I (line 30).
2. Push relevant registers onto the stack
3. load address of ISR into PC (hard coded vector which can be accessed by asserting the decoder, see MicroStore).
4. Fetch.

At this point we execute the ISA instructions for the ISR. RTS is the last instruction executed. RTS restores the pushed registers (PC will pickup where it left off), enable interrupts. Enable interrupts may be redundant if interrupts are already enabled, context would be a simple sub-routine, which RTS would also service.

In the event where we do not WTI and just execute say a loop, an interrupt can occur as an external event, forcing our program to branch to the ISR. This context is taken care of. When I is set the JISR would load and thus perform similarly as above.

One point to note: the PC by default will increment by 1during a fetch, when JISR is executed, it must decrement the PC by 1 to ensure the next fetch after a RTS picks up the proper instruction.

Modifications to the MPU will now need to be made. I is an external non scheduled event, which must be blocked if interrupts are disabled. As well, when we disable interrupts, we can simultaneously clear I in the PSW. The logic is provided by the context of use. ‘I’, will only be disabled if we are about to service an interrupt (JISR), and thus are already in the JISR. Create a cct, which will latch a disable interrupt signal from the MS, and clear this latched disable when an enable signal from the MS is received. When the disable I is set, it should also force I in the PSW to 0. The output of I from the PSW, is fed into the MS unit as well as the 8 bit Y to select IR or JISR.

(Part C)

Output of the system is accomplished by writing information to a register at 0xC0. Create a register which is addressed as 0xC0. The data lines of the register should drive 2, 7 segment displays (4 bits per display). Included in the MPU resources is a “**4 to 7**” display driver.

Input is a 1 byte register (8 bit Sensor, manual input) where the data lines are appropriately buffered so we can read them (like a memory read), and only when addresses 0xC4. As part of the control mechanism, a push button beside the Sensor, will send the interrupt. So..... we enter a value into the sensor, hit return, then press button to send the interrupt.

(Part D)

As part of the final project you will be required to make a 15 min appointment with your instructor for demonstration of your MPU. Email your instructor to set up a time.

To receive a mark, you will be required to present to your instructor 1 of the following ISA level programs. Should your program execute successfully, you will receive a mark appropriate for the level of difficulty the program exhibited, which should be a reflection of project completeness and correctness.

### Rubric for mark assignment

|  |  |  |
| --- | --- | --- |
| Mark | Description | Result |
| 0 | Did Not present | Very happy instructor, no work involved, Very unhappy student. |
| 10-20 | Based on the level of failure to get anything running. | Instructor will grumble, become very quiet, exhibit agitation, will retreat to his office to think things over. Don’t go here. |
| 30 | Write a program which will compute the following Sum = X + (Y) – (Z), where X is an immediate, Sum,Y and Z are memory locations. Test with X=7, Y=12, Z=B | Instructor may ask for different values of X,Y and Z. Student should be able to show the instructor the value of Sum. Mark of 30 or less. Likely a failed project. |
| 50 | Write a program to compute the Sum of N numbers. Program is to immediately store N in a memory location N, then compute sum of N storing the partial and final sum in location N+1. | Instructor may ask for several values of N to be demonstrated. Mark of 50 or less. |
| 65 | Same as above, except a subroutine is to be coded which adds the current value of N to the sum. Thus, 1 call to the subroutine for every partial summation. N and Sum are global variable. | Instructor may ask for several values of N to be demonstrated. Mark of 65 or less. It will be expected that a student at the very least can complete the 50% bracket if attempting this bracket. |
| 75 | Same as above, implementing a recursive subroutine call to perform the summation. Assume the running sum is a global variable, but N is a parameter which is pushed onto the stack during call set up. | Instructor will insist N is pushed onto the stack, and that the next sub-routine call will test N for base case condition. A mark of 75 if working to the standard set out. At the very least the student should demonstrate, above bracket should things not work out. |
| 85 | Write a program which will accept 2-8 bit numbers from the input. Call a subroutine which will produce the sum of all numbers between the 2 just entered (assume they are entered low – hi). The sum can be a signed 16 bit number which should be displayed. The result is to be displayed on the LCD at 0xC0 and a 2nd LCD placed at 0xC1.  | Instructor will assign a mark in the range 50 to 85 depending on program execution. |
| 100 | Write a program which will start incrementing a memory location “Sum” from 0 by a factor of 0x0A. When the program detects an interrupt, it is to retrieve the input and subtract it from “Sum”. The program is to Hlt when an arithmetic overflow is detected.The value of Sum is constantly displayed to the LCD output. | Instructor will assign a mark in the range of 50 to 100 depending on the proper execution of program. |

## Submission Instructions

1. Package your entire project into a zip file.
2. Be sure to include the micro code spread sheet, Excel file.
3. Celebrate the completion of Cosc 3p92!