**Battle Tanks/Turtles Teacher’s Notes**

Thank to Pete Dring for the original idea which can be found at:

<http://pddring.github.io/cpu-battle-tank/>

The Modified Version that these notes and other related resources support, can be found at:

<http://uvhs-comp.github.io/cpu-battle-tank/>

The purpose of these notes and supporting documentation and resources is to enable Teachers unfamiliar with the concept of low level programming, or else are wishing to find an engaging way to introducing the concept to Key Stage 3 students.

**About Battle Tanks**

The idea behind the battle tanks program is it looks like a traditional computer game, where the player has to control a Tank around a playing area to achieve a particular outcome or set of Outcomes. The difference is, that instead of the player controlling the tank using the mouse or keys, the tank needs to be issued with computer instructions. These computer instructions must be written in assembly language

Any CPU can only work with very simple instructions to carry out any programming task. Battle tanks is a way of creating instructions for a simulated CPU which produces a visual result for the student. To program the CPU you must use **Assembly language** which a very low level set of instructions to work with. Assembly language is often even more difficult to comprehend at first than higher level languages e.g. Visual Basic, Python, Smallbasic. Battle tanks gives you an insight into what is going on inside the CPU when you write a program to achieve a particular outcome.

**Low Level Language v High Level Language**

It you were using a high level language to program the tank you would probably use commands such as:

**Tank.MoveForward(2**) # Moves tank 2 spaces forward

**Tank.MoveBackward(5)** # Moves tank 2 spaces forward

**Tank.Turn(180)** #Turn Tank 180 degrees

**Tank.Fire()** # Fire the gun

.. and so on.

However in Battle tanks, the control of the Tank is through instructions written in its own style of Assembly Language, which has a very limited set of instructions compared to a typical High Level Language.

### How The Battle Tanks CPU Works

The way this CPU simulator works is that it has a **register** (storage location) called the **Accumulator** that the user can load values into. These values can then be **Output** to one of four different **channels**, each of which controls the tank in a different way. Instructions to control the tank are written in Assembly language, which is then compiled into machine code which the CPU can understand and execute.

### Output Channels

Below are the different Outputs channel number that must be used to control the tank..

Channel 1 - Movement

Channel 2 - Steering

Channel 3 - Turret Movement

Channel 4 - Firing the Gun

To fully control the tank, each channel does something different depending on what number is Output to the channel. This is summarised below:

|  |  |
| --- | --- |
| **Channel 1 - Movement** | |
| **0** | **1** |
| **Moves Tank Forward** | **Moves Tank Backward** |

So if channel 1 receives a 0 the tank will move forward, if it receives a 1 the tank will move backwards.

|  |  |  |  |
| --- | --- | --- | --- |
| **Channel 2 - Steering** | | | |
| **0** | **2** | **4** | **6** |
| **Turns Tank to Face**  **North** | **Turns Tank to Face**  **East** | **Turns Tank to Face**  **South** | **Turns Tank to Face**  **West** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Channel 3 - Aim Gun** | | | | | | | |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **Turns Tank to Face**  **North** | **Turns Tank to Face**  **North East** | **Turns Tank to Face**  **East** | **Turns Tank to Face**  **South East** | **Turns Tank to Face**  **South** | **Turns Tank to Face**  **South West** | **Turns Tank to Face**  **West** | **Turns Tank to Face**  **North West** |

|  |
| --- |
| **Channel 4 - Firing** |
| **0** |
| Fire Gun |

To Control the tank you need to decide what you want the tanks to do with each instruction eg. move, turn, fire and create Assembly language instructions for each step. To complete each instruction, you will have to decide what you want the tank to do, and then load that instruction it the accumulator, the data loaded into the accumulator can then be output to the appropriate output channel.e.g.

**LDA 1** // **L**oad the (**D**ata) number **1** into the **A**ccumulator

**OUT 1**  // Output the contents of accumulator to Channel 1 (Movement)

**HLT** - Stop the program

### Data Labels

To make writing programs a little easier the programmer can use labels to assign values so that the assembly language is easier to read and the intention of the instruction easier to follow. Comments (instructions ignored when the code is compiled) must be prefaced by a double slash //

// this will be ignored by the compiler.

### Writing Assembly Language Programs

The basic principle is to decide what you want to do e.g move the tank forwards, load that number (0) into the accumulator and then output the contents of the accumulator to the appropriate output channel (channel 1 for movement)

A first Assembly Language program to move the tank **forwards 2 spaces**

|  |
| --- |
| // Move Forwards  LDA Forwards // //uses the label Forwards to load the value 0 into the Accumulator  OUT 1 // Move forwards by Outputting value 0 from accumulator to Output Channel 1  OUT 1 // Move forwards by Outputting value 0 from accumulator to Output Channel 1  HLT // stop the CPU  //Data Labels - help to simplify code by attaching descriptions to the data values  Forwards DAT 0 // assign the value 0 in the Label **Forwards**  Backwards DAT 1 //assign the value 0 in the Label **Backwards** |

Notice that there is no simple way to move more than one space at time, except repeat the Output instruction. This is one feature of low level languages, it often requires more instructions to achieve something that is relatively simple with higher level languages.

An Assembly Language program to move the tank **forwards then** **backwards 2 spaces**

|  |
| --- |
| // Move Forwards  LDA Forwards //uses the label Forwards to load the value 0 into the Accumulator  OUT 1 // Move forwards by Outputting value 0 from accumulator to Output Channel 1  OUT 1 // Move forwards by Outputting value 0 from accumulator to Output Channel 1  //Move Backwards  LDA Backwards //uses the label Backwards to load the value 1 into the Accumulator  OUT 1 // Move Backwards  OUT 1 //// Move forwards by Outputting value 1 from accumulator to Output Channel 1  HLT // stop the CPU  //Data Labels - help to simplify code by attaching descriptions to the data values  Forwards DAT 0 // assign the value 0 in the Label Forwards  Backwards DAT 1 //assign the value 1 in the Label Backwards |

### Turning the tank

The tank can be steered in any one of 4 directions which correspond to compass points. (see the table above.

A simple assembly language program to get the tank to turn right (East):

|  |
| --- |
| //Turn Right  LDA East //uses the label East to load the value 2 into the **Accumulator**  OUT 2 //Turns the tank to the right (East) by outputting the contents of the **Accumulator** to Channel 2  HLT  //Data Labels - help to simplify code by attaching descriptions to the data values  North DAT 0 // assign the value 0 to the Label North  East DAT 2 //assign the value 0 to the Label East  South DAT 4 // assign the value 0 to the Label South  West DAT 6 // assign the value 0 to the Label West |

### Combining Movement and Turning.

E.g. Move the tank 2 spaces forwards Turn West and Move 2 spaces Forward

|  |
| --- |
| // Move Forwards  LDA Forwards  OUT 1  OUT 1  //Turn Tank  LDA West  OUT 2  // Move Forwards  LDA Forwards  OUT 1  OUT 1  //Data Labels - help to simplify code by attaching descriptions to the data values  North DAT 0 // assign the value 0 to the Label North  East DAT 2 //assign the value 0 to the Label East  South DAT 4 // assign the value 0 to the Label South  West DAT 6 // assign the value 0 to the Label West  Forwards DAT 0 // assign the value 0 in the Label Forwards  Backwards DAT 1 //assign the value 0 in the Label Backwards |