Practical 05

Assembly Language

THIS IS A PROCTORED PRACTICAL

YOU MUST SHARE YOUR SCREEN SO YOUR PARTICIPATION IN THIS PRACTICAL CAN FULLY INVIGILATED

1. Create a Github repository "Assembly_and_C"
2. Create a sub directory PRACTICAL_##
3. Add Github link to CA Spreadsheet
   e.g https://STUDENTID.github.com/Assembly_and_c/PRACTICAL_##
4. Invite Lab Supervisors including **MuddyGames** as a collaborators
5. Go to designated group to complete practical
6. Upload completed Practical files to Github repository

NOTE: Use of EASy68K editor and emulator allowed, use of internet allowed, use of slide deck(s) allowed. Installer located here http://www.easy68k.com/

Create a unique file ***e.g. practical_##_part#.X68*** for each practical section below.

Objective Understand and utilise Conditional Branches and Control Structures**:**

1	Create a new 68K project and name the file ***practical_05_part1.X68*** Edit compile and execute the code across and observe while debugging and contents of data registers D0 and D1. Examine and note contents of status registers and. Review questions, what is the purpose of CMP, BNE, BEQ and the Status Register. How many times does GAME_LOOP execute?	 **Source Code Image (click here)**
2	Create a new 68K project	

<table>
<tr>
<td></td>
<td>and name the file
practical_05_part2.X68

Edit compile and execute the code across and observe while debugging and contents of memory, data registers and address registers.

Modify the code such that the COLLISION Branch in executed when the player and enemy are at the same X and Y coordinates 55 and 55

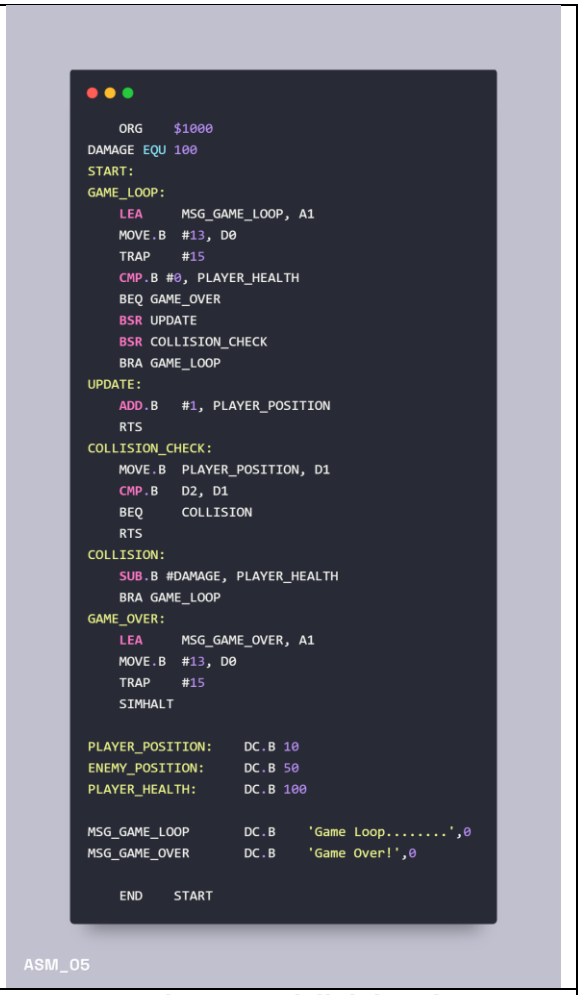
Load Player and Enemy X and Y Positions from the Arrays PLAYER_POSITION and ENEMY_POSITION

Review questions, what do the BEQ instructions mean, what Addressing Modes are used?</td>
<td>

```
    ORG    $1000
START:
    LEA PLAYER_POSITION, A1
    LEA ENEMY_POSITION, A2

    MOVE.B #50, D1
    MOVE.B #0, D2

COLLISION_CHECK:
    CMP D2, D1
    BLT NEXT_MOVE
    BEQ COLLISION

NEXT_MOVE:
    ADD.B #1, D2
    BRA COLLISION_CHECK

COLLISION:
    BRA GAME_OVER

GAME_OVER:
    SIMHALT

PLAYER_POSITION: DC.B 10, 15
ENEMY_POSITION: DC.B 50, 55

    END    START
```

ASM_05

Source Code Image (click here)
</td>
</tr>
<tr>
<td>**3**</td>
<td>Create a new 68K project</td>
<td></td>
</tr>
</table>

	and name the file ***practical_05_part3.X68*** Edit compile and execute the code across and observe while debugging and contents of memory, data registers and address registers. Review questions, what are EQU, BEQ, LEA, TRAP #15, D0 used for within this code. How are MEMORY locations used. Register instructions mean, what Addressing Modes are used?	 ASM_05 **Source Code Image (click here)**
4	Create a new 68K project	

```
    ORG    $1000
DAMAGE EQU 100
START:
GAME_LOOP:
    LEA     MSG_GAME_LOOP, A1
    MOVE.B  #13, D0
    TRAP    #15
    CMP.B #0, PLAYER_HEALTH
    BEQ GAME_OVER
    BSR UPDATE
    BSR COLLISION_CHECK
    BRA GAME_LOOP
UPDATE:
    ADD.B   #1, PLAYER_POSITION
    RTS
COLLISION_CHECK:
    MOVE.B  PLAYER_POSITION, D1
    CMP.B   D2, D1
    BEQ     COLLISION
    RTS
COLLISION:
    SUB.B #DAMAGE, PLAYER_HEALTH
    BRA GAME_LOOP
GAME_OVER:
    LEA     MSG_GAME_OVER, A1
    MOVE.B  #13, D0
    TRAP    #15
    SIMHALT

PLAYER_POSITION:    DC.B 10
ENEMY_POSITION:     DC.B 50
PLAYER_HEALTH:      DC.B 100

MSG_GAME_LOOP       DC.B    'Game Loop........',0
MSG_GAME_OVER       DC.B    'Game Over!',0

    END    START
```

<table>
<tr>
<td></td>
<td>and name the file
practical_05_part4.X68

Edit compile and execute the code across and observe while debugging and contents of memory.

Examine and note contents of address registers and memory.

Review questions, what are the Branch Instructions useful for BLE and BEQ, what Addressing Modes are used?

See FLOW DIAGRAM</td>
<td>

```
    ORG    $1000

; D0 Used for Trap Codes, An Counting ENEMIES
; D1 Used for Players Health
; D2 Used for Enemy Positions
; D4 used for ENEMY count
; A0 for counting enemies
; A1 for progress messages
; A2 used for PLAYER_X
; A3 used for ENEMY_POSITIONS

DAMAGE          EQU 10              ; Damage when Collisions occur

START:
    LEA         PLAYER_X, A2        ; Player will move along X axis
    LEA         ENEMY_POSITIONS, A3 ; Enemy positions or spawn points
    BRA         GAME_LOOP

GAME_LOOP:
    BSR         COUNT_ENEMIES       ; Size of the ENEMY_POSITIONS Array
    MOVE.B      D0, D4              ; Reset Enemy count at begining of loop

    LEA         MSG_GAMELOOP, A1    ; Load and Draw Gameloop Message
    BSR         DRAW

    BSR         UPDATE              ; Call UPDATE
    BRA         GAME_LOOP           ; Call GAME_LOOP

UPDATE:

    MOVE.B      PLAYER_HEALTH, D1   ; Move Player Health into D1

    LEA         MSG_HEALTH, A1      ; Load and Draw Health Message
    BSR         DRAW

    MOVE.B      #3, D0              ; Display signed number in D1.L in decimal in smallest field.
    TRAP        #15

    LEA         CRLF, A1            ; Load and Carriage Return and Line Feed
    BSR         DRAW

    CMP         #0, D1              ; Check if Health is 0
    BLE         GAME_OVER           ; Game Over
    ADD.L       #1, (A2)            ; Change X position of Player
    BSR         COLLISION           ; Check for Collisions with ENEMIES
    RTS

DRAW:
    MOVE.B      #13, D0
    TRAP        #15
    RTS

COLLISION:
    MOVE.B      #1, D5              ; Loop through Array
    LEA         ENEMY_POSITIONS, A3 ; Load ENEMY_POSITIONS Array

ITERATE:
    CMP         D5, D4              ; Counter for ENEMY_POSITIONS
    ADD.B       #1, D5              ; Subtract one
    BEQ         GAME_LOOP           ; If 0 goto GAME_LOOP
    BNE         CHECK_ENEMIES       ; If not Check for Enemy Collisions
    RTS

CHECK_ENEMIES:
    MOVE.L      (A3)+, D2           ; Move ENEMIES position to D2
    MOVE.L      (A2), D3            ; Move Player position to D3

    CMP.L       D2, D3              ; Compare Player and Enemy position
    BEQ         REDUCE_HEALTH       ; If Equal reduce Health
    BNE         ITERATE             ; Iterate to next Enemy

REDUCE_HEALTH:
    LEA         MSG_COLLISION, A1   ; Load and Draw Collision Message
    BSR         DRAW

    SUB.B       #DAMAGE, PLAYER_HEALTH

    LEA         MSG_DAMAGE, A1      ; Load and Draw Damage Message
    BSR         DRAW

    BRA         GAME_LOOP

GAME_OVER:
    LEA         MSG_GAMEOVER, A1
    BSR         DRAW
    SIMHALT

COUNT_ENEMIES:
    MOVE.B      #0, D0              ; Initialize counter to 0
    LEA         ENEMY_POSITIONS, A0 ; Load the address of the array into A0

COUNT_ENEMIES_LOOP:
    TST.L       (A0)+               ; Test the byte at the address in A0 (Check if 0) and increment A0
    BEQ         END_COUNTING        ; If the byte is zero, it's the end of the array

    ADD.B       #1, D0              ; Increment the counter
    BRA         COUNT_ENEMIES_LOOP  ; Repeat the loop

END_COUNTING:
    RTS

PLAYER_X            DC.L    1
PLAYER_HEALTH       DC.B    100
MSG_HEALTH          DC.B    'Health : ', 0
MSG_GAMELOOP        DC.B    'Game Loop', 0
MSG_COLLISION       DC.B    'Collision', 0
MSG_GAMEOVER        DC.B    'Game Over', 0
MSG_DAMAGE          DC.B    'Taking Damage', 0
CRLF                DC.B    $0D,$0A,0
ENEMY_POSITIONS     DC.L    20, 30, 40, 50, 60, 70, 80, 90, 200, 300, 500, 0 ; Terminate Array with a 0

    END    START
```

ASM_05

</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Source Code Image (click here)]</td>
</tr>
<tr>
<td>**5**</td>
<td>Create a new 68K project and designate the file as ***practical_05_part5.X68***.

Review questions, what is the instruction TST.L useful for and what Addressing Modes are used?</td>
<td>**4 Specification**: Using Part 4 modify the code as follows;

• Add a **POWER_UPS** see Fig 5.1.1 Array
These are locations where when player is at this position, they receive a Health **POWER_UPS**

• Modify the Code so that the player receives PO when these positions</td>
</tr>
</table>

		are reached
		
		• Modify the code, to improve maintainability by using the VARIABLE Memory Locations (as in Part 4)
		
		PLAYER_X
		PLAYER_HEALTH
		
		e.g.
		
		From
		`ADD.L #1, (A2)`
		
		TO
		`ADD.L #1, PLAYER_X`

`POWER_UPS  DC.L  10, 55, 220, 0 ; Terminate Array with a 0` ***Figure 5.1.1***		
6	Complete Practical Quiz which will be provided by Lab Supervisor	

Demonstrate completed assembly files at the end of the LAB and ensure it has been checked

Student Name		**Student Number**	
Date		**Checked**	

Practical 05

Assembly Language

FLOW DIAGRAM