Computer Organization and Assembly Languages

Final Project—PONG game in GBA

資工三 B94902047 王政堯

資工二 B95902045 李曜成

資工二 B95902044 石穎

目錄：

一. 前言介紹：

PONG game 這個遊戲相信大家一定都不陌生，甚至有許多人稱之爲所有遊戲中的「開創始組」，所有的遊戲都是從這款遊戲開始慢慢轉變發展而來的，在這次「計算機與組合語言」的 Final Project 中，我們在 GBA 的平台(事實上是一個模擬器)上，實際實作了一個 PONG game，在實做的過程中我們才終於了解爲何它被稱之爲所有遊戲的根本。因爲它包含了一個遊戲基本的組成要素，首先是「背景」(background)，再來就是各個「object」(在 GBA 中我們通常稱之爲 sprite)，而且也包含了「AI」的部分，也就是由電腦控制 object 使之與使用者互動的部分，這些都是組成一個遊戲最重要的要素，透過這些要素的彼此互動加上評分系統，使得遊戲有輸家、有贏家，仔細想想，每個遊戲不就是這樣嗎？

當然不可否認的是，在實作的過程我們也發現，即使是這麼「基本」的小遊戲，我們也就遇到許多問題，更不用說是一些更有趣、更難的遊戲了，但是，透過實作 PONG game 來了解遊戲中各個基本要素的組成、特性，我想這是每個想實作遊戲的程式者一個很好的開始，那接下來我們就分幾個段落來介紹我們此次的 project(由於整個程式並沒有很龐大複雜，所以我們盡量不把 Code 的內容放到報告中)

二. GBA 硬體介紹:

要做一個遊戲之前，最重要的事是要決定要在哪個平台或是模擬器上來實作，每個平台都有著不同的特性，例如：記憶體大小、CPU 處理速度、畫面刷新速率、最高解析度……等等，都隨著不同的平台而有所不同。在我們這次 project 中選擇用 GBA(因爲涵蓋在上課的內容中)，所以首先必須先介紹一些關於 GBA 的硬體知識。在此我們僅列出與我們 project 中較相關的部分，其餘部分在參考資料的連結或網路上的許多教學中皆有更詳細的介紹，爲了方便起見，我們大部分使用英文。

Video

- 240x160 pixel, 15bit color LCD screen. The original GBA screen was not backlit, but the SP's and Micro's are.
- 3 bitmap modes and 3 tilemap modes and sprites.
- 4 individual tilemap layers (backgrounds) and 128 sprites (objects).
- Affine transformations (rotate/scale/shear) on 2 backgrounds and 32 objects.
- Special graphic effects: mosaic, additive blend, fade to white/black.

Sound

- 6 channels total
- 4 tone generators from the original GameBoy: 2 square wave, 1 general wave and one noise generator.
- 2 'DirectSound' channels for playing samples and music.

Miscellaneous

- 10 buttons (or keys): 4-way directional pad, Select/Start, fire buttons A/B, shoulder buttons L/R.
- 14 hardware interrupts.
- 4-player multiplayer mode via a multiboot cable.
- Optional infrared, solar and gyroscopic interfaces. Other interfaces have also been made by some.
- Main programming platforms: C/C++ and assembly, though there are tools for Pascal, Forth, LUA and others as well. Easy to start with, yet hard to truly master.

Memory Sections

This section lists the various memory areas. It's basically a summary of the GBATek section on memory.

area	start	end	length	port-size	description
System ROM	0000:0000h	0000:3FFF	16kb	32 bit	BIOS memory. You can execute it, but not read it (i.o.w, touch, don't look).
EWRAM	0200:0000h	0203:FFFF	256kb	16 bit	External work RAM. Is available for your code and data. If you're using a multiboot cable, this is where the downloaded code goes and execution starts (normally execution starts at ROM). Due to the 16bit port, you want this section's code to be THUMB code.
IWRAM	0300:0000h	0300:7FFFF	32kb	32 bit	This is also available for code and data. The 32-bit bus and the fact that it's embedded in the CPU make this the fastest memory section. The 32bit bus means that ARM instructions can be loaded at once, so put your ARM code here.
IO RAM	0400:0000h	0401:03FF	1kb	16 bit	Memory-mapped IO registers. These have nothing to do with the CPU registers you use in assembly so the name can be a bit confusing. Don't blame me for that. This section is where you control graphics, sound,

					buttons and other features.
PAL RAM	0500:0000h	0500:03FFh	1kb	16 bit	Memory for two palettes containing 256 entries of 15-bit colors each. The first is for backgrounds, the second for sprites.
VRAM	0600:0000h	0601:7FFFF	96kb	16 bit	Video RAM. This is where the data used for backgrounds and sprites are stored. The interpretation of this data depends on a number of things, including video mode and background and sprite settings.
OAM	0700:0000h	0700:03FFh	1kb	32 bit	Object Attribute Memory. This is where you control the sprites.
PAK ROM	0800:0000h	var	var	16 bit	Game Pak ROM. This is where the game is located and execution starts, except when you're running from a multiboot cable. The size is variable, but the limit is 32 MB. It's a 16bit bus, so THUMB code is preferable over ARM code here.
Cart RAM	0e00:0000h	var	var	8 bit	This is where saved data is stored. Cart RAM can be in the form of SRAM, Flash ROM or EEPROM. Programmatically they all do the same thing: store data. The total size is

					variable, but 64kb is a good indication.

以上資料來自：http://www.coranac.com/tonc/text/hardware.htm

而有關畫面刷新的速率如下圖

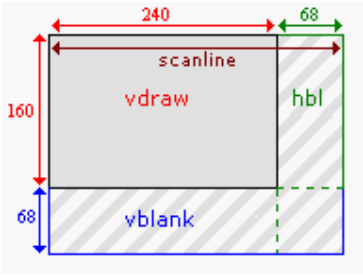

Fig 4.1: vdraw, vblank and hblank periods.

subject	length	cycles
pixel	1	4
HDraw	240px	960
HBlank	68px	272
scanline	Hdraw+Hbl	1232
VDraw	160*scanline	197120
VBlank	68*scanline	83776
refresh	VDraw+Vbl	280896

Table 4.1: Display timing details

在對 GBA 的硬體有些了解後，我們緊接著就進入我們這次的 project 主題—PONG game

三. PONG game

- **Background**

在 GBA 中的背景(background)有分兩種，一種是 **Bitmap Background**，另一種是 **Tiles map Background**。這兩種主要的差別在於組成背景的元素不同，Bitmap Background 中每一個資料代表著一個 pixel 的值，也就是該 pixel 所要呈現的顏色，而 Tile map Background 中每一個資料則是代表一個 tile index 分別對應到不同的 tile，另外一項差異則是在於執行速度，因爲 Bitmap Background 是由軟體「告訴」硬體，讓硬體對每個畫面中的 pixel 填上正確的值，是屬於「軟體著色」，而 Tile map Background 因爲 GBA 的硬體有支援，所以是屬於「硬體著色」，因此，執行速度上，對於同一張 background，用 tile map 的方式通常比 bitmap 的分式來的有效率。

GBA 中控制畫面 Display 主要跟 REG_DISPCNT, REG_DISPSTAT and REG_VCOUNT 這三個 register 有關。

Display registers: REG_DISPCNT, REG_DISPSTAT and REG_VCOUNT

REG_DISPCNT @ 0400:0000h													
F	E	D	C	B	A	9	8	7	6	5	4	3	2 1 0
OW	W1	W0	Obj	BG3	BG2	BG1	BG0	FB	OM	HB	PS	GB	Mode

bits	name	define	description
0-2	Mode	DCNT_MODEx. *DCNT_MODE#*	Sets video mode. 0, 1, 2 are tiled modes; 3, 4, 5 are bitmap modes.
3	GB	DCNT_GB	Is set if cartridge is a GBC game. Read-only.
4	PS	DCNT_PAGE	Page select. Modes 4 and 5 can use page flipping for smoother animation. This bit selects the displayed page (and allowing the other one to be drawn on without artifacts).
5	HB	DCNT_OAM_HBL	Allows access to OAM in an HBlank. OAM is normally locked in VDraw. Will reduce the amount of sprite pixels rendered per line.
6	OM	DCNT_OBJ_1D	Object mapping mode. Tile memory can be seen as a 32x32 matrix of tiles. When sprites are composed of multiple tiles high, this bit tells whether the next row of tiles lies beneath the previous, in correspondence with the matrix structure (2D mapping, OM=0), or right next to it, so that memory is arranged as an array of sprites (1D mapping OM=1). More on this in the sprite chapter.
7	FB	DCNT_BLANK	Force a screen blank.
8-B	BG0-BG3, Obj	DCNT_BGx, DCNT_OBJ. *DCNT_LAYER#*	Enables rendering of the corresponding background and sprites.
D-F	W0-OW	DCNT_WINx, DCNT_WINOBJ	Enables the use of windows 0, 1 and Object window, respectively. Windows can be used to mask out certain areas (like the lamp did in Zelda:LTTP).

REG_DISPSTAT @ 0400:0004h							
F E D C B A 9 8	7 6	5	4	3	2	1	0
VcT	-	VcI	HbI	VbI	VcS	HbS	VbS

bits	name	define	description
0	VbS	DSTAT_IN_VBL	VBlank status, read only. Will be set inside VBlank, clear in VDraw.
1	HbS	DSTAT_IN_HBL	HBlank status, read only. Will be set inside HBlank.
2	VcS	DSTAT_IN_VCT	VCount trigger status. Set if the current scanline matches the scanline trigger (REG_VCOUNT == REG_DISPSTAT{8-F})
3	VbI	DSTAT_VBL_IRQ	VBlank interrupt request. If set, an interrupt will be fired at VBlank.
4	HbI	DSTAT_HBL_IRQ	HBlank interrupt request.
5	VcI	DSTAT_VCT_IRQ	VCount interrupt request. Fires interrupt if current scanline matches trigger value.
8-F	VcT	*DSTAT_VCT#*	VCount trigger value. If the current scanline is at this value, bit 2 is set and an interrupt is fired if requested.

REG_VCOUNT @ 0400:0006h (read-only)	
F E D C B A 9 8	7 6 5 4 3 2 1 0
-	Vc

bits	name	description
0-7	Vc	Vertical count. Range is [0,227]

這次我們的 PONG game 有兩張背景，都是用 tile map 的方式，而且每個 tile 的大小是 8x8 pixels(GBA tile 中最基本的大小)，而如何產生這個 tile map 資料則是透過 Mappy 這套軟體來實作，最後再由 Mappy 輸出 GBA 所能接受的格式。

GBA 中要實作 Tile map Background，必須對以下兩個暫存器設定特定的值

REG_BGxCNT @ 0400:0008 + 2*x*

F E	D	C B A 9 8	7	6	5 4	3 2	1 0
Sz	Wr	SBB	CM	Mos	-	CBB	Pr

bits	name	define	description
0-1	Pr	*BG_PRIO#*	**Priority**. Determines drawing order of backgrounds.
2-3	CBB	*BG_CBB#*	**Character Base Block**. Sets the charblock that serves as the base for character/tile indexing. Values: 0-3.
6	Mos	BG_MOSAIC	**Mosaic** flag. Enables mosaic effect.
7	CM	BG_4BPP, BG_8BPP	**Color Mode**. 16 colors (4bpp) if cleared; 256 colors (8bpp) if set.
8-C	SBB	*BG_SBB#*	**Screen Base Block**. Sets the screenblock that serves as the base for screen-entry/map indexing. Values: 0-31.
D	Wr	BG_WRAP	**Affine Wrapping** flag. If set, affine background wrap around at their edges. Has no effect on regular backgrounds as they wrap around by default.
E-F	Sz	*BG_SIZE#*, *see below*	**Background Size**. Regular and affine backgrounds have different sizes available to them. The sizes, in tiles and in pixels, can be found in table 9.5.

Screen entry format for regular backgrounds

F E D C	B	A	9 8 7 6 5 4 3 2 1 0
PB	VF	HF	TID

bits	name	define	description
0-9	TID	*SE_ID#*	**Tile-index** of the SE.
A-B	HF, VF	SE_HFLIP, SE_VFLIP. *SE_FLIP#*	**Horizontal/vertical flipping** flags.
C-F	PB	*SE_PALBANK#*	**Palette bank** to use when in 16-color mode. Has no effect for 256-color bgs (REG_BGxCNT{6} is set).

- **Sprite**

Sprite，簡單來說，相信每個人都玩過「瑪莉歐」這個經典的遊戲，在瑪莉歐遊戲畫面中的瑪莉歐就是一個 sprite，當然頭目酷巴也是 sprite 而且是個比較大的 sprite。在我們的 PONG game 中主要有三種 sprite：球(Ball)、板子(Bar)、評

分分數(number0-9)，其中球又分爲普通球、火球。而爲了讓畫面能顯示出 Sprite，就像顯示出 tile map background 一樣，我們必須把 Sprite data 存入適當的記憶體位址，再把 sprite 的「狀態」(例如：畫面 X,Y 座標、Color 模式)設定適當的暫存器。

OBJ_ATTR.attr0															
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
Sh		CM	Mos	GM		OM		Y							

bits	name	define	description
0-7	Y	*ATTR0_Y#*	**Y coordinate**. Marks the top of the sprite.
8-9	OM	ATTR0_REG, ATTR0_AFF, ATTR0_HIDE, ATTR0_AFF_DBL. *ATTR0_MODE#*	**(Affine) object mode**. Use to hide the sprite or govern affine mode. **00**. Normal rendering. **01**. Sprite is an affine sprite, using affine matrix specified by attr1{9-D} **10**. Disables rendering (hides the sprite) **11**. Affine sprite using double rendering area. See affine sprites for more.
A-B	GM	ATTR0_BLEND, ATTR0_WIN. *ATTR0_GFX#*	**Gfx mode**. Flags for special effects. **00**. Normal rendering. **01**. Enables alpha blending. Covered here. **10**. Object is part of the object window. The sprite itself isn't rendered, but serves as a mask for bgs and other sprites. (I think, haven't used it yet) **11**. Forbidden.
C	Mos	ATTR0_MOSAIC	Enables mosaic effect. Covered here.
D	CM	ATTR0_4BPP, ATTR0_8BPP	**Color mode**. 16 colors (4bpp) if cleared; 256 colors (8bpp) if set.

E-F	Sh	ATTR0_SQUARE, ATTR0_WIDE, ATTR0_TALL. *ATTR0_SHAPE#*	**Sprite shape**. This and the sprite's size (`attr1{E-F}`) determines the sprite's real size, see table 8.4.

OBJ_ATTR.attr1

F E	D	C	B A 9	8 7 6 5 4 3 2 1 0
Sz	VF	HF	-	X
-	AID			-

bits	name	define	description
0-8	X	*ATTR1_X#*	**X coordinate**. Marks left of the sprite.
9-D	AID	*ATTR1_AFF#*	**Affine index**. Specifies the OAM_AFF_ENTY this sprite uses. Valid *only* if the affine flag (`attr0{8}`) is set.
C-D	HF, VF	ATTR1_HFLIP, ATTR1_VFLIP. *ATTR1_FLIP#*	**Horizontal/vertical flipping** flags. Used *only* if the affine flag (`attr0`) is clear; otherwise they're part of the affine index.
E-F	Sz	*ATTR1_SIZE#*	**Sprite size**. Kinda. Together with the shape bits (`attr0{E-F}`) these determine the sprite's real size, see table 8.4.

OBJ_ATTR.attr2

F E D C	B A	9 8 7 6 5 4 3 2 1 0
PB	Pr	TID

bits	name		description
0-9	TID	*ATTR2_ID#*	Base **tile-index** of sprite. Note that in bitmap modes this must be 512 or higher.
A-B	Pr	*ATTR2_PRIO#*	**Priority**. Higher priorities are drawn first (and therefore can be covered by later sprites and backgrounds). Sprites cover backgrounds of the same priority, and for sprites of the same priority, the higher OBJ_ATTRs are drawn first.
C-F	PB	*ATTR2_PALBANK#*	**Palette-bank** to use when in 16-color mode. Has no effect if the color mode flag (`attr0{C}`) is set.

- **AI**

人工智慧(AI)。相信每個人都有過這樣的經驗：在「洛克人」遊戲中嘗試了好幾次卻仍然打不過某個強力的 BOSS、在「極速快感」中不管玩幾次都無法擠進前三名、或者「西洋棋」無論思考多久都贏不了電腦。在這些情況中有一部份的因素就是在於控制這個 BOSS 的 AI 真是太強了！一個遊戲中 AI 的部分掌握了整個遊戲的難度，進而影響了整個遊戲的「有趣性」。畢竟，沒有人喜歡玩太簡單沒有挑戰性或是太難而一直卡關的遊戲，所以如何調整適當的 AI 難度，寫出相對應的 Code 對遊戲中式個非常重要的環節。

在我們 PONG game 遊戲中，AI 扮演的角色就是與我們玩家對抗的那塊板子(bar)，AI 會「盡量」去接住玩家所彈過來的球，不讓玩家輕易得分。我們實作 AI 的流程可以分成下面幾個步驟：首先，實作出一個完美、不會輸的 AI，之後透過讓 AI 的移動速度變慢、球的速度變快的方法使 AI 變的不完美，簡單來說就是「變笨」，但是這樣還不夠，我們最後利用 rand()使得 AI 在對球做出反應的時間不固定，試著去模擬出就像一個玩家在操控一樣。在這些過程中，許多的設定都是經過不斷地微調、嘗試，才模擬出最後的結果。

- **Collision Detection**

有關遊戲中物件(object or sprite)的碰撞，主要有三種演算法：範圍偵測、顏色偵測、行進路線偵測。考慮這次 project PONG game 的實際需求，行進路線不大適合，然而雖然顏色偵測是三種中能最準確地判斷出兩個霧建是否真的有發生碰撞，但是需要一個 Mask 的圖層與遇預測的 project 作 AND，所以時作上會出現較多困難，加上 PONG game 的物件並不多，且都是規則狀的物件，所以我們到最後選擇用範圍偵測。

而我們所用的範圍偵測是屬於「矩形偵測」，也就是將每個 sprite 都視爲是一個矩形，而因爲我們知道 sprite 在畫面中的(X,Y)座標，也知道每個 sprite 的大小，所以透過簡單的數學式及可判斷出兩個 sprite 筆次是否有碰撞，雖然這不像顏色偵測或行進路線偵測來得準確，但是卻是最快最好實作的一個方法。

- **Speed Up**

由於在此遊戲中除了 AI 之外，最主要的部份就是 Sprite 的移動(板子、球)，因此在遊戲執行的過程中，絕大多數的時間是在把圖片顯示在螢幕上，也就是把 Sprite 的資料複製到 OAM(Object Attribute Memory)上，所以我們想利用 Assembly 來對這一部份做加速。於是我們用 Arm 寫了一個 BlockCopy ((unsigned int*) to, (char *) from, size_t size)的函式，直接對連續的資料讀取，省去了用 array 的 index

來讀取的麻煩。其次，我們在 BlockCopy 中應用了 loop unrolling 的概念，運用八個暫存器(r3-r10)，一次讀取 32bytes，來減低迴圈的次數。有了這一個想法，不只是關於 OAM 的存取，包括一些連續、大量的資料，也都可以用這個函式來達成。

BlockCopy.s

```
@ BlockCopy:      r0: to, r1: from, r3: size
.text
.align      2
.global     BlockCopy
.type       BlockCopy, %function
BlockCopy:
    sub       r2, r2, #16
    ldmia     r1!, {r3-r10}
    stmia     r0!, {r3-r10}
    cmp       r2, #0
    bgt       BlockCopy
    mov       pc, lr
.size BlockCopy, .-BlockCopy
.ident      "GCC: (GNU) 3.3.2"
```

- Play Game

接下來簡單說明一下我們這次 Pong game 的遊戲守則，規則就如大家所知，球會在玩家與 AI 的板子之間不斷來回碰撞，若玩家沒有順利接到球，則玩家就輸一分，相反地，若 AI 的板子沒接到球，則就是玩家得一分，一局總共十分，先拿到十分的人獲勝。而按鍵操作方面，當不論輸一分或得一分，之後畫面會先停住，此時按下 A 鍵即可繼續下一球；那我們將原本的 Pong game 做了小延伸，也就是「火球」，當玩家按下 B 鍵的時候，板子會發光，此時球碰到板子後就會變成火球，火球的球速比原來的球還快，遊戲難度會提升，而此時若玩家按下 L 鍵，則可以回復成原來的板子，此時球若到板子，就可以從火球恢復成原來的球，簡單來說，玩家可以自由的操縱球的快慢，增加遊戲的有趣性，但是別忘了，AI 可不是省油的燈，當玩家分數領先 AI 很多的時候，AI 也會同樣的用火球來提升球速，所以簡單一句話，**Enjoy the game!**

四. 結論與心得

這是我第一次實作有關遊戲設計的 project，雖然很辛苦，但是其實也很有趣(尤其是把 AI 一步一步變笨的時候)，其實一開始很希望能做出一個「不錯」的 AVG game 或 RPG game 但是因爲實力、時間的限制，所以後來決定做 PONG game 這個所有遊戲的「始祖」，並試著去加入一些新的元素然而，即使是這麼一個基本的遊戲，在實做過程都出現許多問題，也許是我 Code 的功力還不足，不過透過這次的 project，讓我學到了有關遊戲設計許多的背景知識，也了解到遊戲設計中團隊分工、合作的重要，相信下次一定可以做出更棒的遊戲的。(By 王政堯)

這次的經驗讓我體會到接觸一些平常不常接觸的事物所帶給我的困難以及障礙。在這一次的專題中實做的方式是以 C 以及 Arm 作爲基礎，在做這個專題之前，我自己可能覺得對這些東西有一定的熟悉程度，不過當它們同時結合並呈現在不同的領域(GBA)上時，事情就完全不是這麼一回事了。感謝這一次的機會讓我學到不少東西，也很高興能與其他人一起合作完成這次專題，希望以後有機會還能再接觸此類的專題。(By 李曜成)

很開心有機會能參與這次的 project！以前玩的 GBA，沒想到現在可以自己動手做遊戲，也經由這次機會讓我更了解 GBA。雖然這是我們第一次做遊戲，剛開始難免不知所措，但我們花了一些時間認識 GBA 相關背景知識(比如說 sprite 跟 backgrond)，也了解到 ARM 應用範圍的廣泛（之前總認爲組語只要學 intel 就好了，何必學 ARM）。另外經過這次經驗，發現不管是做遊戲或是寫軟體，事前的準備很重要，就像剛開始我們花不少時間在了解 sprite 還有一些 mapping mode。也很感謝同組的夥伴，總之這次的經驗很新鮮也學到不少，也有那麼一絲「學以致用」的成就感。(By 石穎)

五. 參考資料

1. 遊戲設計概論 胡昭民著 博碩文化股份有限公司 西元 2007 年 9 月

2. http://www.coranac.com/tonc/text/toc.htm
   裡面有 GBA 較詳細的硬體知識，例如：暫存器的 bit 功用

3. http://theharbourfamily.com/jonathan/?page_id=89
   內有詳細的 PDF 教學檔可以下載

4. http://www.gsarchives.net/index2.php?category=all&system=gameboyadvance&letter=all

在這裡找了一些需要的 Sprite(雖然後來都沒用到)

5. http://www.gamespp.com/tutorials/ConvertingArtToSprites.html
   在這了解 Sprite 要怎麼去做

6. http://www.vgmaps.com/Atlas/GBA/index.htm
   有相當多的遊戲 background(Bitmap)

7. http://www.tilemap.co.uk/mappy.php
   project 中製作 tile map 用到的軟體 Mappy