Kick Off 3 - GameBoy Development

There are two (2) major technical difficulties to overcome in translating this game from the S-NES to the GameBoy:

i) THE MACRO LANGUAGE

It was hoped the macro language would be a proper macro system. Unfortunately, the designer and/or developer of the langauge has done nothing more than RENAME the 658 micro-processor instructions.

The first version of the system supplied to me ASSUMED a great many things about the way micro-processors work. Indeed, it appears that the designer believes FLAGs registers on all micro-processors are affected in the same way. For example:

```
    LDA #0    : on 6502-based processors (like the processor
                in the S-NES) would set the ZERO FLAG to 0.
                It's equivalent Z80 instruction (the GameBoy
                uses a Z80-based processor) "LD A,0" leaves
                the ZERO FLAG unaffected.
```

To duplicate LDA #0 on Z80-based processors, we would need to write the following:

```
    LD  A,0
    AND A
```

Now the accumulator holds the desired value (in this case 0), and the ZERO FLAG is now set to 0. But now we discover that the CARRY FLAG is different on both 6502 and Z80-based processors. On the Z80-based machine, the CARRY FLAG is now 0. On the 6502-based machine, the CARRY FLAG has been unaffected by the operation of the instruction and has retained it's former value. The design error is still present: the designer has done nothing more than RENAME the 658 micro-processor instructions.

The problem becomes more acute when it is realised that the designer believes all ASSEMBLY LANGUAGES work in the same way. This is another FATAL ASSUMPTION, leading to ADDRESS MODE conflicts when attempting to translate the macros to other micro-processors.

For example:

```
    LDA8 macro
    ISM8
    LDA \1
    endm
```

The designer has RENAMED the 6502 LDA instruction to LDA8, but he has not taken into account the difference between IMMEDIATE addressing and MEMORY addressing. This problem in 6502 ASSEMBLY LANGUAGE is resolved by the type of operand following the instruction.

For example:

```
    LDA #0 : this loads the accumulator with an IMMEDIATE value, in
             this case "0"

    LDA 0  : this loads the accumulator from a MEMORY location, in
             this case from the memory location $0000, which could
             contain any value from "0" to "255"
```

The ASSEMBLY LANGUAGE would be able to resolve the ADDRESS MODE simply by the presence or the non-presence of the ASSEMBLER DIRECTIVE # (hash sign). While some other ASSEMBLY LANGUAGES use similar notation (the Mega Drive 68000 assembler for instance), the Z80 ASSMEBLY LANGUAGE uses quite a different syntax. For example:

```
         6502          Z80

        LDA #0        LD A,0
        LDA 0         LD A,(0)
```

Due to lack of vision on the part of the designer, it is not possible to translate this instruction into Z80 as LDA \1 could mean either IMMEDIATE or MEMORY addressing depending on the presence of the hash sign. The macro system is therefore AMBIGUOUS in its choice of syntax.

There is further conflict between which NUMBER BASE is being represented in 6502. For example:

```
    LDA $1234 : this will load the accumulator with the value
                stored at memory location $1234.
```

Our ASSEMBLY LANGUAGE would object to this, citing a bad label as $1234 means the LOCAL LABEL called $1234, and not the memory location $1234.

Also LDA #13 on 6502-based systems means load the accumulator with the value 13. Our assembler interprets the # (hash sign) as meaning HEXADECIMAL. Our accumulator would now hold the value 19. Again, the designer of the macro system has ASSUMED a great many things about the way micro-processors and assembly languages work.

In order for a macro language to work on ALL machines, it must FREE itself from processor and assembler syntax. The system supplied to me, simply because it's designer has done nothing more than RENAME 658 instructions, will forever be chained, bound and gagged to its 6502 heritage.

The second version of the macro system supplied to me resolved the IMMEDIATE / MEMORY addressing conflict, but the syntax and FLAGS problems remain.

I have described the problems associated with just one (1) instruction. The 6502 has 55 others, each with similar problems, and IMMEDIATE and MEMORY addressing are but 2 of 8 ways in which the processor can address data.

Most of the design flaws in the macro system can be worked around, but only with laboriously long-winded translation code. For example:

LDA \1,Y becomes:

```
        LD  HL,Y
        LD  E,(HL)
        LD  D,0
        LD  HL,\1
        ADD HL,DE
        LD  A,(HL)
        LD  (ACCUMULATOR),A
        INC HL
        LD  A,(HL)
        LD  (ACCUMULATOR+1),A

        ; + code to update the FLAGS
```

The Z80 version of the S-NES code will be 6 to 15 times bigger - and therefor 6 to 15 times slower - than the original code. What we are having to do is EMULATE the 6502 instruction set rather than translate a language. We are, in effect, trying to teach GERMANS how to speak FRENCH, but both the GERMANS and ourselves have to learn ICELANDIC before we can even begin to talk to each other.

ii) BANK SIZE

On the Game Boy, our bank size is 16K - half that of the S-NES. Given that our code is now 6 to 15 times bigger than S-NES code, it is very difficult to make things fit into banks without having to chop code to pieces.

Our very first task is to install the S-NES code routine by routine, ensure it fits within the 16K banks, and make sure that the rest of the code knows where we've put it. Unlike the Mega Drive which doesn't have banks (just one large chunk of memory) we need to do this chopping before anything substantial will run at all.

We have also a major problem with the transformation of world coordinates to 3D pitch coordinates. The table which accomplishes this on the S-NES is 116K - half the available Game Boy memory. On a S-NES, 116K is just a drop in the ocean, an extravagant use of memory. We must find some way of achieving the same effect on the Game Boy without resorting to massive tables. Until this is done, we can make no headway with positioning sprites on screen.

WORK COMPLETED

1) The menu system
2) Pitch scroll
3) Translation of macro system
4) Inclusion of ball logic