

Saint Seiya Ōgon densetsu kanketsu-hen



Tạm dịch: Saint Seiya - Truyền thuyết Hoàng kim, phần kết.

Thể loại: side scrolling, RPG

Hệ máy: Family Computer hay còn gọi Famicom (FC) hoặc Nintendo Entertainment System (NES).

Số người chơi: 01

Hãng phát triển: Shinsei

Xuất bản: Bandai

Ngày phát mãi: 30 tháng 5, 1988

1. Khái quát

Saint Seiya Ōgon densetsu kanketsu-hen hay còn được gọi là Saint Seiya 2, là phần cuối cùng trong bộ 2 game về Saint Seiya trên hệ

máy Famicom. Bộ 2 về Saint Seiya này có nội dung dựa trên bộ Anime cùng tên vốn dựa trên nguyên tác Manga của họa sĩ Kurumada Masami với motif về thần thoại Hy Lạp. Khi tà ác tràn lan trên mặt đất thì nữ thần trí tuệ Athena cùng các thiếu niên bảo vệ nàng sẽ xuất hiện. Những thiếu niên này được các chòm sao bảo hộ, thân khoác Thánh y của các chòm sao và có sức mạnh kinh thiên hăi nhân.

Saint Seiya Ōgon densetsu kanketsu-hen kể lại phần nhóm thiếu niên Seiya và nữ thần Athena rời Nhật Bản, sang Hy Lạp để chống lại thế lực tà ác trong Thánh vực. Tại đây, Athena bị mũi tên vàng đâm vào ngực và không ai có thể rút mũi tên ra, ngoại trừ Giáo hoàng. Nếu không dẫn Giáo hoàng tới trong vòng 12 giờ thì Athena sẽ vong mạng. Nhóm Seiya phải cấp rút đột phá 12 cung để tới được phòng Giáo hoàng. Trên đường là các Hoàng kim Thánh đấu sĩ thuộc 12 cung Hoàng đạo cản trở họ...

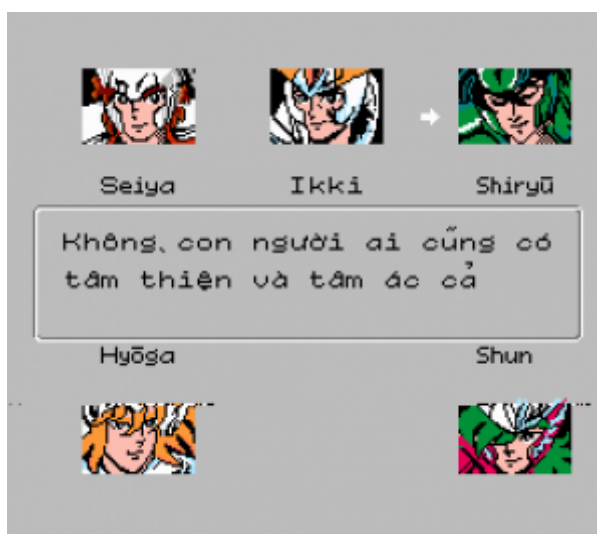
2. Đôi lời cá nhân

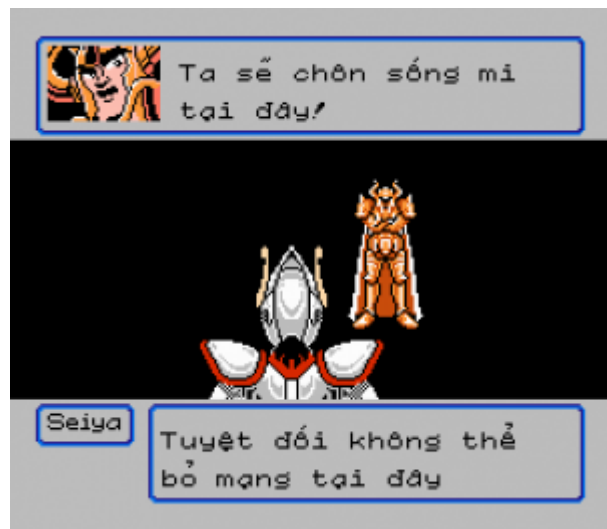
Bản game này ra đời trên hệ máy Famicom vào năm 1988, tính tới nay có lẽ còn nhiều tuổi hơn một số lượng khá đông gamer hiện tại. Nhưng đối với tôi thì nó là một phần của tuổi thơ và có sức hút rất lớn. Tôi bắt gặp băng Saint Seiya 2 này từ năm lớp 4, và chơi miệt mài cho tới năm lớp 7 mới phá đảo nổi. Thời đó không mấy cửa hàng trò chơi điện tử có băng này, vì hầu như không có người chơi vì rất khó, cho nên tôi phải đạp xe đi gần chục cây số để chơi. Thời gian sau này, khi mọi phương tiện đã dễ dàng với phần mềm giả lập trên máy tính và cho đến nay, thỉnh thoảng tôi vẫn chơi lại Saint Seiya Ōgon densetsu kanketsu-hen với nhiều cảm xúc, kỷ niệm khó tả.

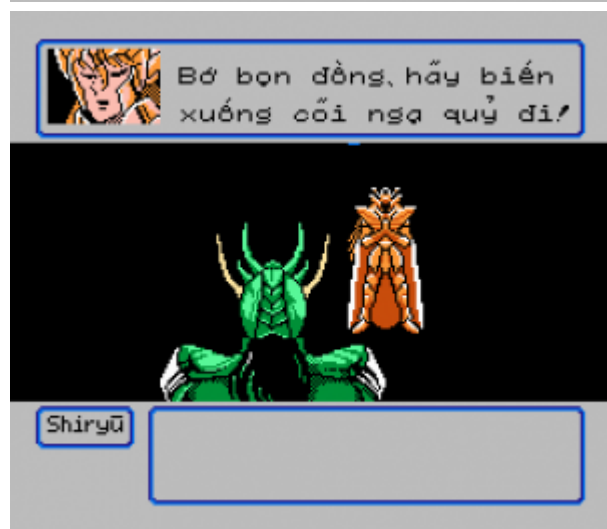
Vì vậy, kể từ khi bước chân vào con đường dịch game từ năm 2009,

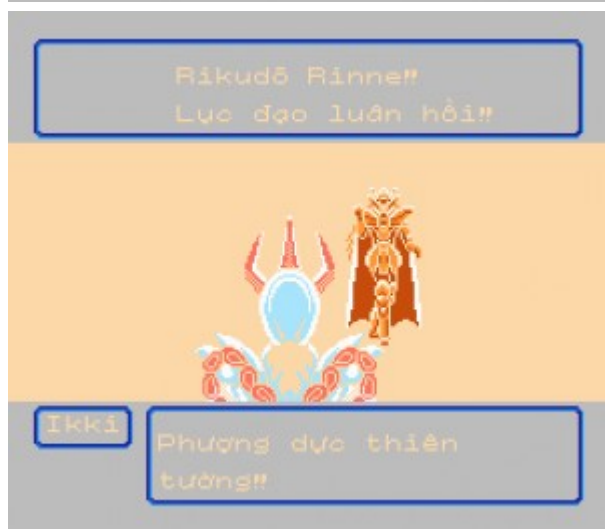
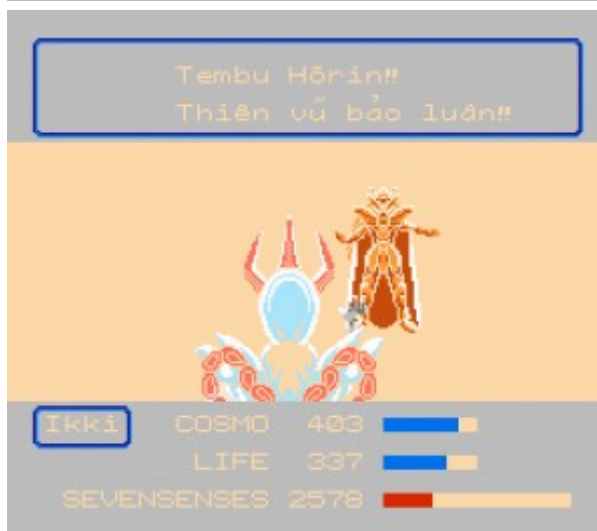
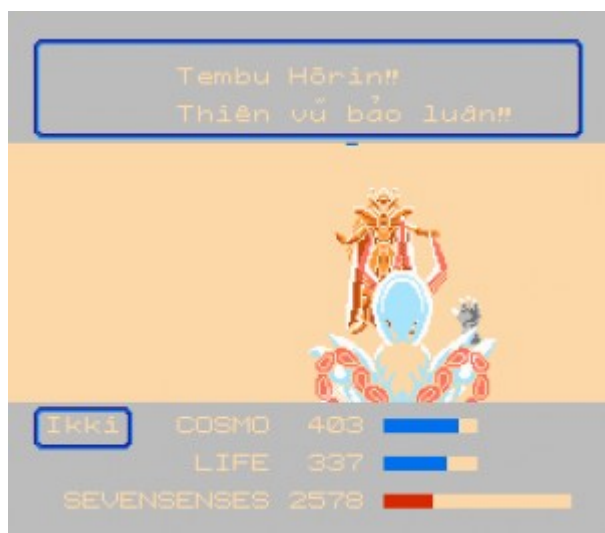
tôi cũng đã ước ao có được một bản dịch Việt ngữ cho Saint Seiya Ōgon densetsu kanketsu-hen, nhưng lực bất tòng tâm. Đương thời, tôi chỉ tập trung vào ngôn ngữ 65816 của hệ máy SNES sau này chứ không tập trung vào 6502 của NES. Game NES so về dung lượng hay độ phức tạp thì không thể sánh bằng các hệ máy sau này như PS, PC, NDS... nhưng về độ khó thì không hề thua kém, thậm chí còn khó hơn. Vì nhiều hạn chế kỹ thuật của phần cứng mà việc bỏ dấu tiếng Việt cho game NES dường như là điều bất khả. Từ trước đến nay chưa hề có một bản dịch tiếng Việt nào cho bất cứ game NES nào.

Nhưng cuối cùng, sau một thời gian miệt mài học tập 6502, ngôn ngữ lập trình của NES thì cuối cùng, tôi cũng thực hiện được mong ước của mình với Saint Seiya Ōgon densetsu kanketsu-hen. Có thể nói, đây là bản dịch Việt ngữ đầu tiên của game NES.









3. Đào lô đồ

Download tại: <http://gokuraku-shujo.blogspot.com/2015/07/SS2.html>

Bản dịch đã được kiểm chứng là chạy tốt trên các giả lập Famicom (NES) cho hệ điều hành Windows như Virtual NES, FCEUX,... và các loại giả lập cho hệ điều hành Android.

Các trang web tải giả lập:

<http://www.virtualnes.com/>

<http://www.fceux.com/web/home.html>

4. Credit

+ Asm65816 VI: hack, dịch.
+ Phần mềm sử dụng: EmEditor (biên tập văn bản), Unikey (gõ tiếng Việt), YY-CHR (tạo font), Windhex32 (view là chính), Cartographer (truy xuất text), Atlas (chèn text đã dịch), FCEUX (giả lập, debugger Famicom), MicroSoft Excel (quản lý linh tinh).

5. Bonus

Phần này hướng dẫn khái quát về việc dịch Saint Seiya Ōgon densetsu kanketsu-hen sang tiếng Việt cho những ai quan tâm.

a. Dễ hay khó?

Đơn giản hay dễ dàng, phức tạp hay khó khăn, đây là những khái niệm hay bị nhầm lẫn với nhau. Có những cái đơn giản nhưng lại rất có, có những cái tuy phức tạp nhưng lại rất dễ. Tôi nhớ năm nào, có một danh họa tranh thủy mặc Trung Quốc sang thăm một trường ĐH mỹ thuật Việt Nam, vốn thiên về tranh sơn dầu phương Tây. Danh họa thủy mặc này biểu diễn trước hội chúng, lấy giấy bút vẽ nguệch

ngoạc vài nét và rao bán với giá vài chục nghìn USD. Đám sinh viên trở mắt, hỏi các danh họa Tây phương vẽ cầu kỳ, chi tiết từ năm này sang tháng nọ mà chưa chắc đã bán được với giá đó, có sao ông hý hoáy vài nét đơn giản lại bán đắt cổ vậy? Danh họa thủy mặc kia đáp: các anh có biết, để hý hoáy được vài nét như vậy thì tôi phải luyện cả đời.

Cũng vậy, nếu so về dung lượng hay độ phức tạp, nếu phần lớn game PC và các hệ console hiện tại như PS3, PS4 có dung lượng của một con bò thì game PSX ngày xưa có kích cỡ một con mèo, game SNES là một con bọ trong khi game NES chỉ là một con vi trùng.

Kích thước nhỏ hơn rất nhiều, số lượng tập lệnh ít hơn và đơn giản hơn so với các hệ máy đàn em, nhưng chính vì nhỏ hơn, đơn giản hơn nên việc hack/dịch game NES gặp phải nhiều vấn đề khó khăn hơn, bên cạnh những cái dễ dàng hơn. Khó khăn như thế nào, sẽ được lần lượt trình bày ở những phần bên dưới.

Nếu như có chút ít kiến thức về đặc trưng phần cứng của máy NES, cũng như kiến thức về ngôn ngữ 6502 của nó thì mọi việc sẽ bớt khó đi rất nhiều.

b. Chuẩn bị

Cần có các phần mềm sau để dịch một game NES. Tất cả đều dễ dàng tìm thấy trên Google. Dĩ nhiên danh sách ở đây chỉ là gợi ý.

- + Text editor: có thể dùng Notepad có sẵn trong Windows.
- + Bộ gõ tiếng Việt:
- + Sửa font, hình ảnh: tile layer pro, YY-CHR...
- + Emulator kèm Debugger: xuất sắc nhất cho tới nay là FCEUX.
- + Hex editor: Windhex có chức năng xem chữ Kana với table lập sẵn.
- + Dump text: Cartographer
- + Chèn text: Atlas

c. Cấu trúc Memory map

Trước khi đi vào hack/dịch Saint Seiya ōgon densetsu kanketsu-hen thì ta cần biết khái quát về máy NES. Có thể tham khảo loạt bài lập trình 6502 để biết thêm chi tiết. Dưới đây là tóm lược.

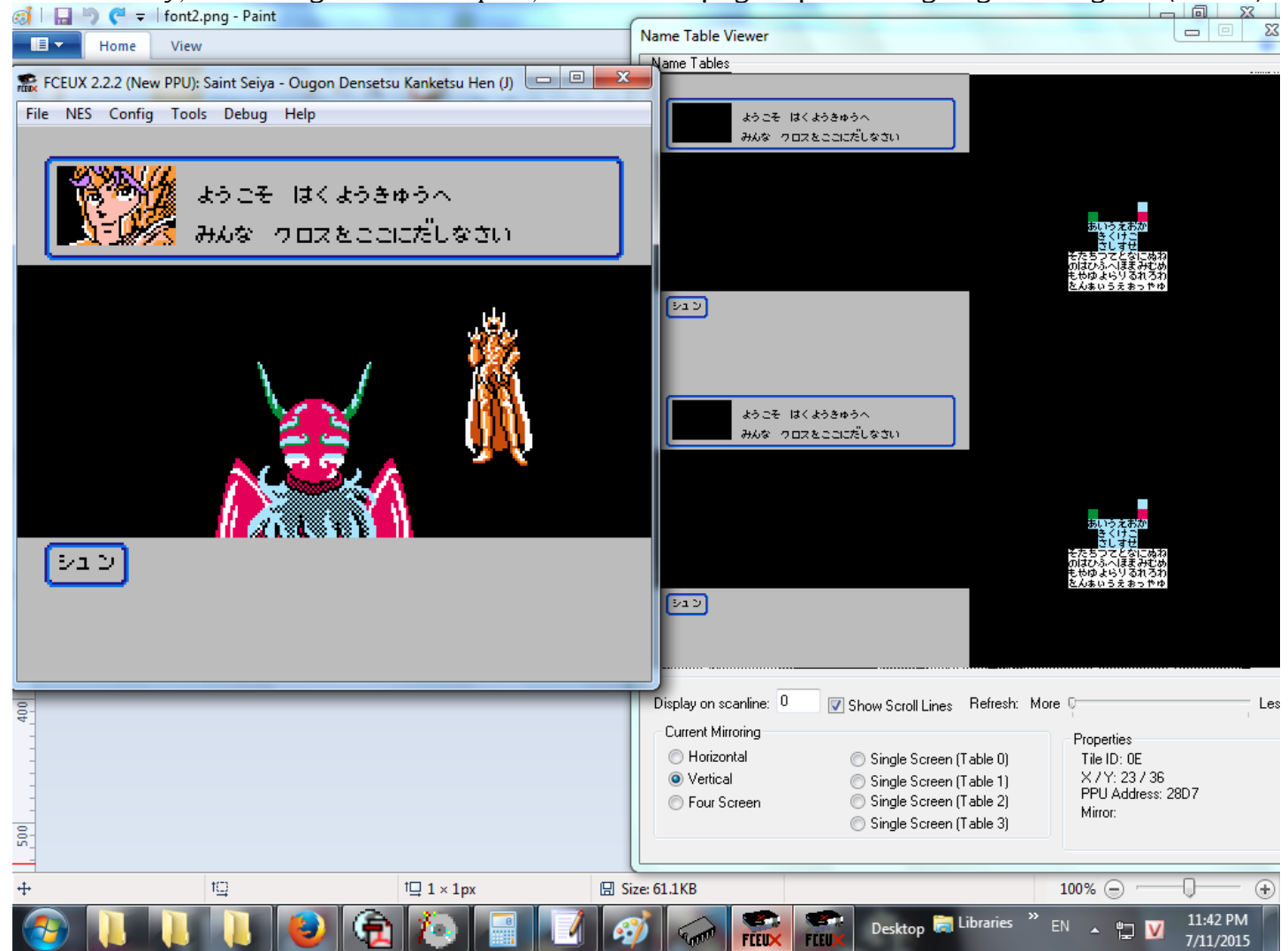
- \$0000-\$07FF: RAM nội bộ, dung lượng 2KB bên trong NES, user tự do sử dụng
- \$0800-\$1FFF: đối xứng gương của RAM
- \$2000-\$2007: các cổng truy cập vào PPU (I/O Register)
- \$2008-\$3FFF: đối xứng gương của I/O Register
- \$4000-\$401F: các cổng truy cập vào APU và Controller
- \$4020-\$5FFF: ROM mở rộng
- \$6000-\$7FFF: WRAM có thể có hay không bên trong Cartridge (backup RAM)
- \$8000-\$BFFF: ROM bên trong Cartridge
- \$C000-\$FFFF: ROM bên trong Cartridge

Lưu ý từ \$00 đến \$7FF là nơi chứa những giá trị thường xuyên được truy cập, chẳng hạn như tọa độ của nhân vật, Cosmo, Life, pointer hội thoại... Máy NES sử dụng các Register 8 bit và có thể đọc địa chỉ trong dải từ \$00 đến \$FFFF (65 Kilo byte). Tuy nhiên, khi mở Rom bằng Hex editor thì ta thấy Rom có dung lượng 258 KB, vượt quá phạm vi đọc của NES. Tuy nhiên máy NES vẫn đọc được, nhờ

vào khái niệm Mapper như sẽ đề cập ở phần sau.

d. Lập table

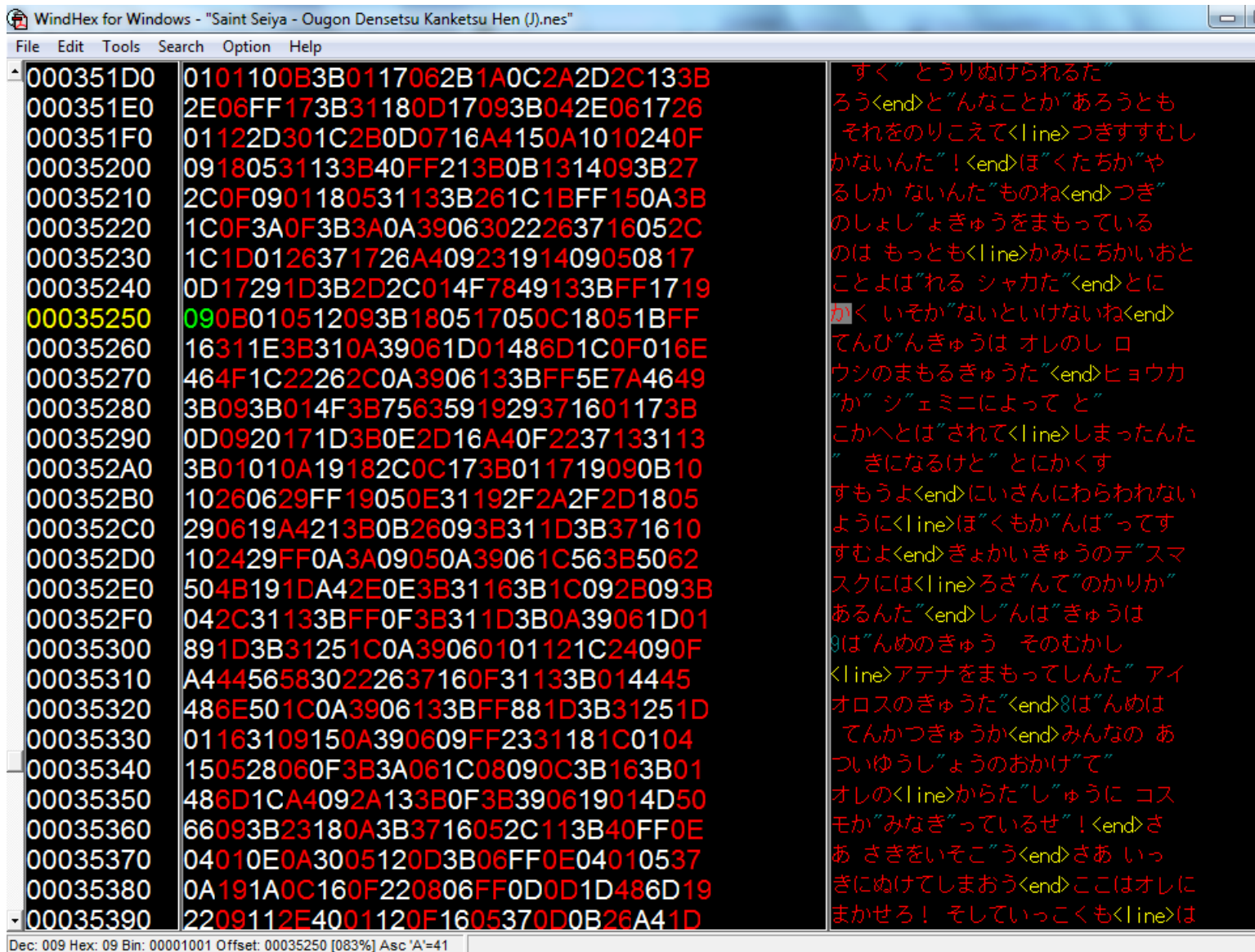
Lập table cho game NES là một việc hết sức đơn giản, nhờ có sự trợ giúp của giả lập FCEUX. Load game bằng FCEUX, chơi đến đoạn có hội thoại, vào mục Debug --> Name table Viewer. Tại đây ta thấy phần thoại cũng được thể hiện trong phần Name tables. rê chuột vào từng ký tự trong màn hình này, nhìn vào góc dưới bên phải, ta sẽ biết được giá trị hex tương ứng của từng chữ (Tile ID).



Phần lớn chữ trong hội thoại của game NES đều ở dạng background (ảnh nền). Name table chính là phần ID (số thứ tự) của từng tile trong thư viện (bank) ảnh nền đó. Bằng cách rê chuột vào từng chữ và nhìn vào góc dưới, ta có thể lập được table dễ dàng. Chẳng hạn:

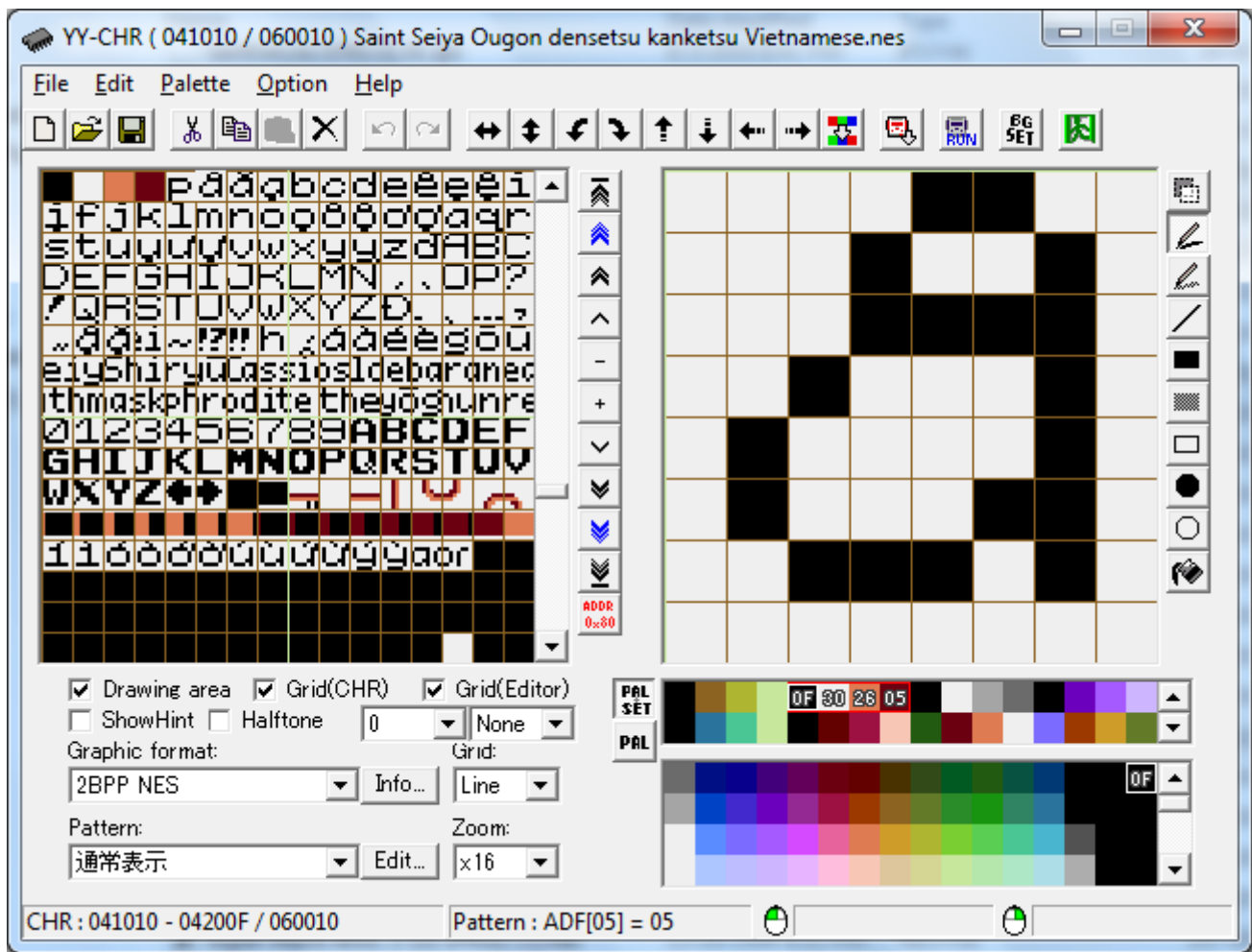
4B=ク
6E=口
50=ス

Sau khi lập xong table hoàn thiện, load ROM bằng Windhex và tham chiếu đến file table vừa lập, ta có thể dò tìm hội thoại nằm ở vị trí nào trong ROM bằng chức năng tìm kiếm chữ Kana (control + K) của Windhex.



e. Chỉnh sửa font

Có thể dùng phần mềm YY-CHR để chỉnh sửa hình ảnh và font chữ trong game. Có thể thấy bộ font bắt đầu từ địa chỉ 0x21050 trở đi.



Như vậy là bó tay rồi sao? Không hẳn là như vậy.
Bây giờ thử bật bản Rom gốc (tiếng Nhật) bằng giả lập xem sao.



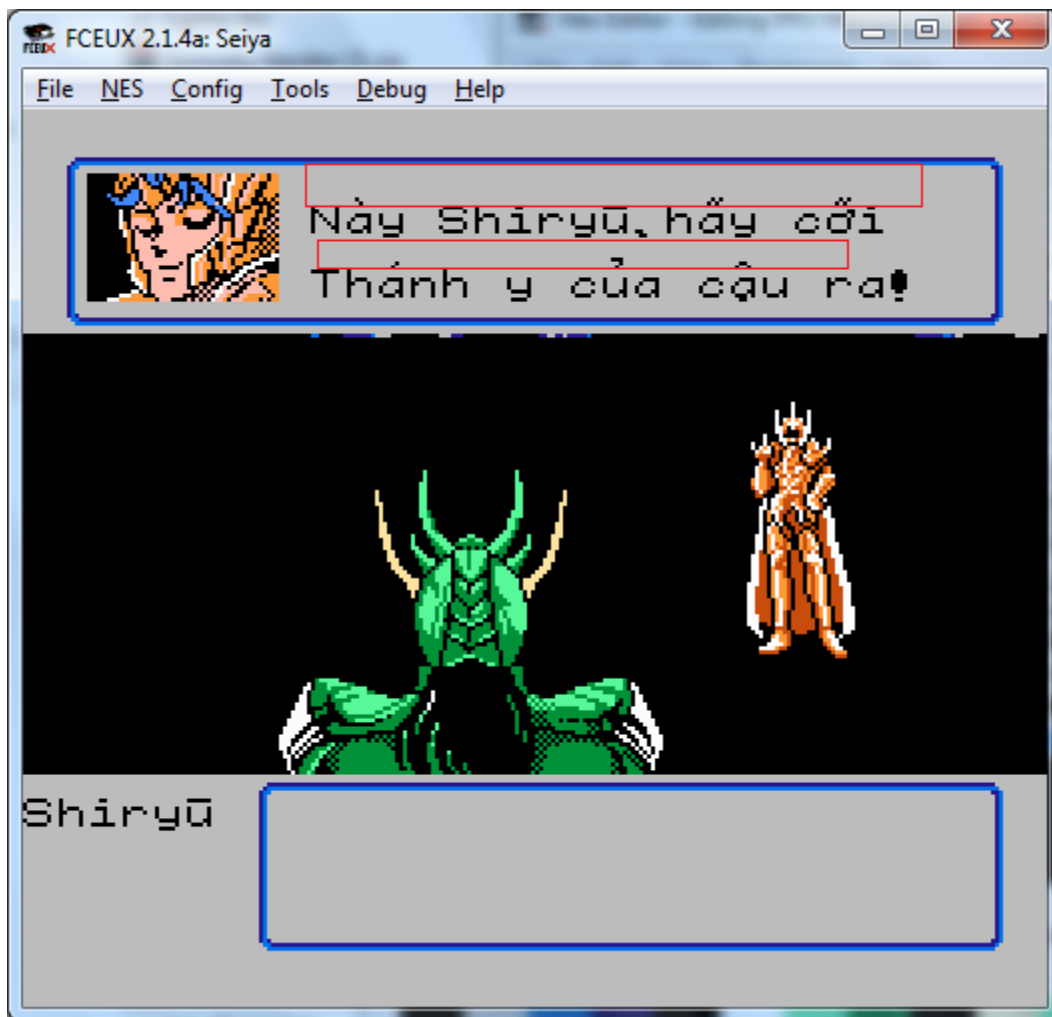
Để ý một chút, ta thấy trong tiếng Nhật có dấu trợ âm và bán trợ âm được viết ngay phía trên một số chữ nhất định. Với tile 8x8 thì dường như nhà sản xuất không thể vẽ được chữ Kana với cả dấu trợ âm, bán trợ âm trong cùng tile. Do vậy họ đã chọn cách viết phần chính vào những hàng chẵn, còn hàng lẻ để viết dấu trợ âm, bán trợ âm vào.

Ta có thể lợi dụng điểm này để thể hiện dấu tiếng Việt, với cùng cách thức như trên.

Nhưng vấn đề là bản Rom tiếng Nhật chỉ có 02 dấu trợ âm, bán trợ âm trong khi tiếng Việt cần nhiều hơn như thế, như "ă", "ã", "ắ", "ằ", "ắ", "ằ", "ắ", "ằ" đã liệt kê bên trên.

Để giải quyết vấn đề này, cần phải tìm ra routine viết chữ ra màn hình của bản tiếng Nhật và copy, nhân rộng số lượng thêm trong bản tiếng Việt. Nói cách khác, kiểu bỏ dấu này hoàn toàn giống với kiểu bỏ dấu VNI WINDOWS của PC, với một ký tự đầy đủ dấu chiếm 2 byte thay vì 1 byte như thông thường.

Sau khi hoàn tất thì sẽ trông như thế này.



Nhưng trước hết phải tìm được routine viết chữ ra màn hình đã.

f. Asm vào cuộc

Từ phần này trở đi sẽ phải vận dụng nhiều kiến thức Asm của NES (6502) để giải quyết. Tuy nhiên, dù có bao nhiêu kiến thức về Asm đi nữa cũng không thể giải quyết được gì nếu thiếu hiểu biết về các địa chỉ RAM của game.

Như đã nói ở phần trên, trong địa chỉ từ \$00 đến \$7FF là nơi chứa những giá trị thường xuyên được truy cập, mỗi game mỗi khác nhau nên cách duy nhất để nắm bắt nó là phải ăn dầm ở đó, thường xuyên nghịch nó để biết nó là cái gì. Đây cũng là một khâu mất nhiều thời gian nhất trong việc hack dịch game. Để hack được nó, phải hiểu thấu nó, phải nắm được nó. Chỉ có cách chạm vào nó, ngẫu nhiên nó, ăn ngủ với nó trong thời gian dài mới biết được nó ra sao, như thế nào. Đây là quy tắc chung của việc hack dịch game, không có ngoại lệ.

Ở phần trên, sau khi lập table thì ta có thể xác định được vị trí của hội thoại trong Rom. Ta dễ dàng tìm được vị trí của block hội thoại bắt đầu từ 0x35010. Khoan đã, CPU của NES chỉ có thể đọc địa chỉ trong khoản \$00 đến \$FFFF, vậy hội thoại bắt đầu từ 0x35010 thì làm sao nó đọc tới được?

Câu trả lời nằm ở Mapper. Để khắc phục hạn chế phạm vi đọc của NES, người ta dùng Mapper để phân chia những Rom có dung lượng lớn hơn FFFF byte (65535 byte trở lên) thành nhiều bank.

Load Rom bằng FCEUX, vào tab Help--> Message Log, ta có những thông tin sau:

PRG ROM: 8 x 16KiB
CHR ROM: 16 x 8KiB
ROM CRC32: 0x9561798d
ROM MD5: 0x2c85dedefc8dd7b49b80774de9793506
Mapper #: 1
Mapper name: MMC1
Mirroring: Horizontal
Battery-backed: No
Trained: No

Những thông tin này cho biết: Saint Seiya ōgon densetsu kanketsu-hen dùng hệ thống Mapper #1 có tên MMC1, sử dụng 8 bank cho phần code chương trình, mỗi bank có dung lượng 16 KB (\$4000) và sử dụng 16 bank để chứa dữ liệu đồ họa (CHR), mỗi bank có dung lượng 8 KB (\$2000).

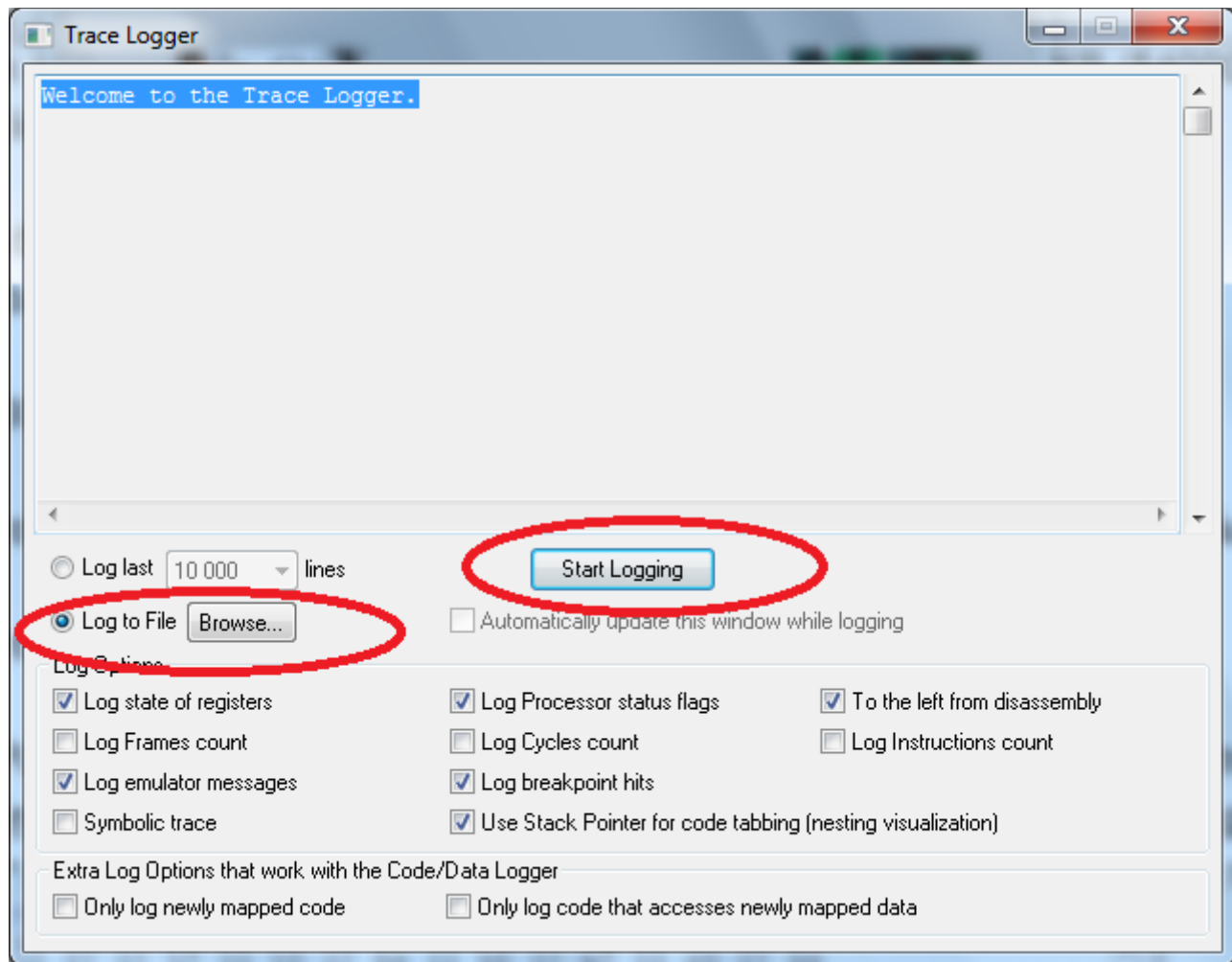
Địa chỉ block hội thoại bắt đầu từ 0x35010, trừ đi \$10 byte (16) header thì là 0x35000.
Vậy, khối hội thoại nằm ở bank \$0D, bắt đầu tại địa chỉ \$1000.

$$35000 - (4000 \times 0D) = 1000$$

Ta đã biết, phần code chương trình bắt đầu từ \$8000 trong CPU. Vậy block hội thoại sẽ bắt đầu tại
 $\$8000 + \$1000 = \$9000$ trong CPU.

Tuy nhiên, \$9000 trong CPU cũng có thể là địa chỉ của 0x5010, 0x9010, 0xD010, 0x11010.... trong ROM. Do vậy cần phải kiểm chứng. Và khi kiểm chứng bằng cách dò tìm giá trị trong CPU (Từ giả lập FCEUX --> Debug --> Hex editor --> View --> Nes Memory) khi hội thoại đang được hiển thị thì ta biết được hội thoại không tồn tại trong \$9000, tức nó không nằm trong phần code PRG. Như vậy chỉ còn một trường hợp xảy ra là Saint Seiya 2 đọc hội thoại từ phần dữ liệu đồ họa CHR. Trong trường hợp này thì không thể dùng cách thông thường để xác định routine hội thoại. Để xác định, phải dùng đến chức năng Trace Logger của Debugger. Bật FCEUX, load game, vào Debug--> Trace Logger

Chọn Log vào file để ghi lại mọi hoạt động của giả lập



Click vào tab "Start Logging" và bắt đầu chơi game tới đoạn có hội thoại. Lưu ý là khi log sẽ ngốn rất nhiều RAM của PC vì mọi câu lệnh thực thi trong giả lập đều được ghi lại.

Mở file log bằng text editor, lần mò từng đoạn code dẫn đến hội thoại, ta thấy có đoạn

```
$8F8E:C9 3B CMP #$3B A:18 X:00 Y:7A S:ED P:nvUbdIzc
$8F90:F0 4F BEQ $8FE1 A:18 X:00 Y:7A S:ED P:NvUbdIzc
$8F92:C9 3C CMP #$3C A:18 X:00 Y:7A S:ED P:NvUbdIzc
$8F94:F0 4B BEQ $8FE1 A:18 X:00 Y:7A S:ED P:NvUbdIzc
$8F96:C9 01 CMP #$01 A:18 X:00 Y:7A S:ED P:NvUbdIzc
$8F98:F0 12 BEQ $8FAC A:18 X:00 Y:7A S:ED P:nvUbdIzc
$8F9A:C9 A4 CMP #$A4 A:18 X:00 Y:7A S:ED P:nvUbdIzc
$8F9C:F0 5F BEQ $8FFD A:18 X:00 Y:7A S:ED P:nvUbdIzc
$8F9E:C9 FF CMP #$FF A:18 X:00 Y:7A S:ED P:nvUbdIzc
$8FA0:D0 17 BNE $8FB9 A:18 X:00 Y:7A S:ED P:nvUbdIzc
```

Nếu đã lập xong table thì ta biết 3B, 3C là giá trị hex của dấu trọc âm và bán trọc âm, A4 là giá trị xuống hàng, 01 là giá trị của khoảng trắng và FF là giá trị kết thúc câu.

Đoạn trên có nghĩa là:

so sánh với 3B

nếu bằng thì nhảy đến địa chỉ \$8FE1

so sánh với 3C

nếu bằng thì nhảy đến địa chỉ \$8FE1
so sánh với 01
nếu bằng thì nhảy đến địa chỉ \$8FAC
so sánh với A4
nếu bằng thì nhảy tới địa chỉ \$8FFD
so sánh với FF
nếu không bằng thì nhảy tới địa chỉ 8FB9

Từ đây ta biết được \$8FE1 trong CPU là địa chỉ bắt đầu routine vẽ dấu trục âm và bán trục âm lên trên một hàng, \$8FAC là địa chỉ bắt đầu routine xuống hàng cho chữ, còn \$8FFD là routine viết chữ bình thường.

Giả sử trong bản font, ta vẫn còn các giá trị trống (chưa gán vào ký tự nào) là B0, B1, B2, B3, B4, B5... và ta định gán các giá trị này cho các dấu: B0=<dấu mũ á>, B1=<dấu mũ ó>, B2=<dấu mũ á, sắc>, B3=<dấu mũ á, huyền>, B4=<dấu mũ á, hỏi>....

thì chỉ cần viết lại như sau:

```
CMP #$B0
BEQ $8FE1
CMP #$B1
BEQ $8FE1
CMP #$B2
BEQ $8FE1
CMP #$B3
BEQ $8FE1
CMP #$B4
BEQ $8FE1
CMP #$B5
BEQ $8FE1
....
```

```
CMP #$01
BEQ $8FAC
CMP #$A4
BEQ $8FFD
CMP #$FF
BNE $8FB9
```

Sau khi viết lại như thế này, giả sử 05=a thì nếu ta chèn vào ROM:

05B3 thì màn hình sẽ hiển thị chữ ầ
05B5 thì màn hình sẽ hiển thị chữ ă
với các dấu nằm ngay bên trên chữ cái.

Tuy nhiên, khi viết lại như thế này thì đoạn code của ta sẽ dài hơn ban đầu. Do vậy phải tìm một không gian trống trong cùng bank để đưa nó vào. Nếu trong bank không còn chỗ trống thì mở rộng Rom và chỉnh lại bank để đưa nó vào.

g. Mở rộng Rom

Mở rộng Rom NES không phải là vấn đề quá khó khăn khi thực hiện thủ công.

Trong \$10 byte header của game NES thì byte thứ 4 chỉ định kích thước của PRG bank. Saint Seiya 2 sử dụng 8 bank PRG và ta có thể mở rộng thành 16 bank, mỗi bank 16 KB. Sau khi mở rộng, chỉ cần thay đổi lại giá trị của byte thứ 4 trong header là được.

Rom gốc ban đầu có 8 bank và vị trí kết thúc của bank thứ 8 là 0x1C010 trong Rom (\$10 byte là header).

$$4000 \times 7 = 1C000 + 10 = 1C010$$

(Bank đầu tiên là bank 0)

Như vậy, ta có thể thêm 8×16 KB vào sau vị trí kết thúc của bank 8, tức là bắt đầu từ 0x1C010 trong ROM. Có thể dùng hex editor để chèn thêm 8 bank, mỗi bank 16KB vào để trở thành 16 bank. Giá trị chèn có thể là 00 hoặc FF.

h. Những vấn đề khác

Một điều dễ thấy nhất là trong số các vấn đề cần phải giải quyết là dung lượng.

Block hội thoại ban đầu là tiếng Nhật, vốn có đặc điểm cô đọng súc tích hơn nên thường ngắn hơn phần dịch tiếng Việt. Hơn nữa, phần tiếng Nhật ít khi sử dụng đến 2 byte để thể hiện 1 ký tự (trợ âm, bán trợ âm) nhưng trong phần dịch tiếng Việt lại sử dụng rất nhiều ký tự có dấu (2 byte), do đó dung lượng của phần dịch tiếng Việt sẽ nhiều hơn phần tiếng Nhật rất nhiều.

Một vấn đề thứ hai là tất cả mọi câu thoại trong tiếng Nhật chỉ cần 2 dòng để hiển thị. Nhưng có những câu khi dịch sang tiếng Việt phải mất 3, 4 dòng. Trong game không có sẵn code để hiển thị dòng thứ 3 (xóa 2 dòng đầu), do đó cần phải viết thêm routine xóa 2 dòng đã hiển thị.

Đối với vấn đề đầu tiên, có thể dùng phương pháp DTE, MTE để thể hiện 2 hay nhiều ký tự bằng 1 byte. Cách làm này giúp tiết kiệm dung lượng rất nhiều.

Còn đối với vấn đề thứ hai, cần phải viết thêm routine. Cả hai hướng giải quyết này đều cần tới khoảng không gian trống để ghi code mới.

Về không gian trống thì ta đã có được bằng cách mở rộng ROM như bên trên. Nhưng để sử dụng được khoảng không gian trống đó, cũng như đặt code mới vào đó, bắt chương trình đọc từ nơi đó thì phải cần đến kiến thức về ASM.