UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CAMPUS APUCARANA

ENGENHARIA DE COMPUTAÇÃO

DANIEL MARTINS DE CARVALHO

GUILHERME CONCEIÇÃO RAMALHO

LEONARDO RAFAEL BUENO

RUAN MATEUS TRIZOTTI

DOCUMENTAÇÃO PARA A CONSTRUÇÃO DE UM EMULADOR DO SUPER NINTENDO ENTERTAINMENT SYSTEM (SNES)

APUCARANA

JUNHO, 2022

DANIEL MARTINS DE CARVALHO

GUILHERME CONCEIÇÃO RAMALHO

LEONARDO RAFAEL BUENO

RUAN MATEUS TRIZOTTI

DOCUMENTAÇÃO PARA A CONSTRUÇÃO DE UM EMULADOR DO SUPER NINTENDO ENTERTAINMENT SYSTEM (SNES)

Relatório elaborado como requisito parcial à obtenção de nota na disciplina de Arquitetura e Organização de Computadores do curso superior de Engenharia de Computação, do Campus Apucarana da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. André Luiz Tinassi D'Amato.

APUCARANA

JUNHO, 2022

RESUMO

O presente relatório tem como objetivo detalhar os métodos e técnicas (em específico os presentes na cpu (WDC 65c816)) necessárias para o funcionamento do Super Nintendo Entertainment System (SNES), abrangendo então, o conjunto e formato de instruções, assim como o conjunto de registradores e seu modo de endereçamento, dispondo ainda um datapath simplificado para melhor compreensão da sua arquitetura.

Palavras chaves: SNES, Registradores, Instruções, Endereçamento e Arquitetura.

SUMÁRIO

1 INTRODUÇÃO

O Super Nintendo Entertainment System (Super NES, SNES, Super nintendo), lançado em 1990 no japão, 1991 nos Estados Unidos, 1992 na Europa e Australásia(oceania) e América do Sul em 1993 (desconsiderando os consoles não oficiais, que já circulavam há algum tempo, em países como o brasil, na América do Sul). É um console de videogame de 16 bits desenvolvido pela Nintendo para ser sucessor do NES (Nintendo Entertainment System ou “nintendinho”), este que possuía apenas 8 bits em sua arquitetura (metade em relação ao super Nintendo). Além de apresentar gráficos e recursos de som melhorados, o SNES superava em muito a maioria dos consoles da época.

O atual relatório visa destacar alguns aspectos do Super Nintendo, como o conjunto e formato das instruções presente em sua construção, o Conjunto de registradores que garantem o funcionamento adequado de todas as operações e instruções, o seu modo de endereçamento e uma breve explicação sobre o seu Data path (em uma forma simplificada).

É importante destacar que para este relatório iremos tomar a CPU 65c816 como CPU principal e fundamental para o Super NES.

Linguagem

A linguagem usada é o 65c816 assembly é a linguagem usada pelo chip Ricoh 5A22 do Super Nintendo Entertainment System (SNES). Dividir as diferentes partes do acrônimo 65c816: 816 significa que o processador pode estar no modo de 8 bits ou no modo de 16 bits. O c significa CMOS, 65 significa que este processador é da família de CPUs 65xx. O processador deve ser bastante revolucionário para a época. Este relatório explica memônicos/instruções (ou seja, opcodes) e como usá-los corretamente. Este relatório não se concentra em tópicos específicos do SNES, como registros de hardware.

Com 65c816 ASM você pode codificar coisas para jogos SNES (como recursos personalizados para Super Mario World). ASM é uma linguagem de programação de 2ª geração, que é de baixo nível em comparação com C#, por exemplo. É um código de máquina legível, que eventualmente é traduzido em código de máquina hexadecimal. Todos os opcodes consistem em 3 letras, juntamente com vários parâmetros.

IDE

Não há IDEs dedicados para montagem 65c816. Você pode usar qualquer editor ASCII, como o Bloco de Notas ou o VS Code. No entanto, algumas pessoas criaram vários plugins para editores de código existentes para adicionar recursos extras, como realce de sintaxe:

- Plugin "65816 Assembly" de Josh Neta para VS Code
- Plugin "65816 SNES Assembly Language Server" da Vice para VS Code.
- Plugin "Asar syntax highlight" do lx5 para o VS Code.
- MatthewCallis' "Suporte à linguagem assembly 65xx" para Atom.

Arquivos de montagem geralmente são salvos com a extensão de arquivo ".asm".

Este tutorial usa a sintaxe que é usada por um montador chamado "Asar", originalmente escrito por Alcaro, agora mantido por vários membros da comunidade SMW Central.

2 CONJUNTO DE INSTRUÇÕES

Figura 1 - Tabela dos conjuntos de instruções do processador w65c816s de 8-16 bits:

1.	ADC	Add Memory to Accumulator with Carry	47.	PHB	Push Data Bank Register on Stack
2.	AND	"AND" Memory with Accumulator	48.	PHD	Push Direct Register on Stack
3.	ASL	Shift One Bit Left, Memory or Accumulator	49.	PHK	Push Program Bank Register on Stack
4.	BCC	Branch on Carry Clear (C=0)	50.	PHP	Push Processor Status on Stack
5.	BCS	Branch on Carry Set (C=1)	51.	PHX	Push Index X on Stack
6.	BEQ	Branch if Equal (Z=1)	52.	PHY	Push Index Y on Stack
7.	BIT	Bit Test	53.	PLA	Pull Accumulator from Stack
8.	BMI	Branch if Result Minus (N=1)	54.	PLB	Pull Data Bank Register from Stack
9.	BNE	Branch if Not Equal (Z=0)	55.	PLD	Pull Direct Register from Stack
10.	BPL	Branch if Result Plus (N=0)	56.	PLP	Pull Processor Status from Stack
11.	BRA	Branch Always	57.	PLX	Pull Index X from Stack
12.	BRK	Force Break	58.	PLY	Pull Index Y from Stack
13.	BRL	Branch Always Long	59.	REP	Reset Status Bits
14.	BVC	Branch on Overflow Clear (V=0)	60.	ROL	Rotate One Bit Left (Memory or Accumulator)
15.	BVS	Branch on Overflow Set (V=1)	61.	ROR	Rotate One Bit Right
16.	CLC	Clear Carry Flag	62.	RTI	Return from Interrupt
17.	CLD	Clear Decimal Mode	63.	RTL	Return from Subroutine Long
18.	CLI	Clear Interrupt Disable Bit	64.	RTS	Return from Subroutine
19.	CLV	Clear Overflow Flag	65.	SBC	Subtract Memory from Accumulator
20.	CMP	Compare Memory and Accumulator	66.	SEP	Set Processor Status Bit
21.	COP	Coprocessor	67.	SEC	Set Carry Flag
22.	CPX	Compare Memory and Index X	68.	SED	Set Decimal Mode
23.	CPY	Compare Memory and Index Y	69.	SEI	Set Interrupt Disable Status
24.	DEC	Decrement Memory or Accumulator by One	70.	STA	Store Accumulator in Memory
25.	DEX	Decrement Index X by One	71.	STP	Stop the Clock
26.	DEY	Decrement Index Y by One	72.	STX	Store Index X in Memory
27.	EOR	"Exclusive OR" Memory with Accumulator	73.	STY	Store Index Y in Memory
28.	INC	Increment Memory or Accumulator by One	74.	STZ	Store Zero in Memory
29.	INX	Increment Index X by One	75.	TAX	Transfer Accumulator in Index X
30.	INY	Increment Index Y by One	76.	TAY	Transfer Accumulator to Index Y
31.	JML	Jump Long	77.	TCD	Transfer C Accumulator to Direct Register
32.	JMP	Jump to New Location	78.	TCS	Transfer C Accumulator to Stack Pointer
33.	JSL	Jump Subroutine Long	79.	TDC	Transfer Direct Register to C Accumulator
34.	JSR	Jump to News Location Saving Return	80.	TRB	Test and Reset Bit
35.	LDA	Load Accumulator with Memory	81.	TSB	Test and Set Bit
36.	LDX	Load Index X with Memory	82.	TSC	Transfer Stack Pointer to C Accumulator
37.	LDY	Load Index Y with Memory	83.	TSX	Transfer Stack Pointer Register to Index X
38.	LSR	Shift One Bit Right (Memory or Accumulator)	84.	TXA	Transfer Index X to Accumulator
39.	MVN	Block Move Negative	85.	TXS	Transfer Index X to Stack Pointer Register
40.	MVP	Block Move Positive	86.	TXY	Transfer Index X to Index Y
41.	NOP	No Operation	87.	TYA	Transfer Index Y to Accumulator
42.	ORA	"OR" Memory with Accumulator	88.	TYX	Transfer Index Y to Index X
43.	PEA	Push Absolute Address	89.	WAI	Wait for Interrupt
44.	PEI	Push Indirect Address	90.	WDM	Reserved for future use
45.	PER	Push Program Counter Relative Address	91.	XBA	Exchange B and A Accumulator
46.	PHA	Push Accumulator on Stack	92.	XCE	Exchange Carry and Emulation Bits

Fonte: The Western Design Center Inc. (2018).

Figura 2 - Tabela com o conjuntos de instrução dos vetores com endereço:

Table 5-2 Emulation Mode Vector Locations (8-bit Mode)

Address	Label	Function
00FFFE,F	IRQB/BRK	Hardware/Software
00FFFC,D	RESETB	Hardware
00FFFA,B	NMIB	Hardware
00FFF8,9	ABORTB	Hardware
00FFF6,7	(Reserved)	Hardware
00FFF4,5	COP	Software
00FFF2,3	(Reserved)	
00FFF0,1	(Reserved)	

Table 5-3 Native Mode Vector Locations (16-bit Mode)

Address	Label	Function
00FFEE,F	IRQB	Hardware
00FFEC,D	(Reserved)	
00FFEA,B	NMIB	Hardware
00FFE8,9	ABORTB	
00FFE6,7	BRK	Software
00FFE4,5	COP	Software
00FFE2,3	(Reserved)	
00FFE0,1	(Reserved)	

Fonte: The Western Design Center Inc. (2018).

2.1. Explicações de Endereço

Esta seção diz respeito à forma como os endereços são escritos, para evitar confusão.

Página Direta

- Por uma questão de simplicidade e explicação, todos os endereços de página direta ("$xx") devem usar o endereço $7E000 como base. Isso deve ser enfatizado nos comentários do bloco de código, bem como nas explicações.

Exemplo:

```
LDA #$42
STA $08          ;Armazene o valor $ 42 no endereço $7E0008
```

Endereçamento Absoluto

- Todos os endereços absolutos ("$xxxx") usam o endereço $7E0000 como base, se o endereço estiver entre $0000-$7FFF inclusive. Para endereços entre $8000-$FFFF, o endereço $000000 deve ser usado como base. Isso deve ser enfatizado nos comentários do bloco de código, bem como nas explicações.

Exemplo:

```
LDA $8002        ;Carregue o valor no endereço $ 008002 em A
STA $1008        ;Armazene esse valor no endereço $ 7E1008
```

2.2. Estilo

Esta seção diz respeito ao estilo do documento.

Endereçamento Absoluto

- Todos os opcodes e instruções devem ser cercados pelo acento grave (`), independentemente do contexto.

Exemplo: O opcode LDA é usado para carregar valores no registrador A. Assim, LDA #$49 carrega o valor $49 em A. Este então pode ser aumentado para o valor $4A usando INC A.

Tabelas de Redução

- Frases e frases dentro das células da tabela geralmente não devem terminar com um ponto.
- Tabelas que introduzem opcodes devem ter os opcodes em negrito e ter pelo menos as três colunas no exemplo abaixo:

Tabela 1 - Tabulação

Código de operação	Nome completo	Explicação
LDA	Carregar no acumulador	Carregar um valor em A

Fonte: Assembly for the SNES.

2.3. Terminologia

Tabela 2 - Uso de certas palavras em determinado contexto.

Regra	Exemplo
Ao se referir ao processador Ricoh 5A22, use o SNES em vez	O SNES é capaz de entrar no modo de 8 bits ou 16 bits
Ao se referir a uma área específica na memória do SNES, sempre coloque o endereço antes, de preferência com a área de memória (ou seja, RAM)	(O) endereço de RAM $ 7E0000 contém [...]
Ao se referir ao acumulador (A) ou aos registradores de índice (X e Y), basta usar A, X ou Y	A está agora no modo de 8 bits. X é usado para indexar endereços
Ao se referir a valores, sempre coloque valor antes dele	A contém o valor $ 00

Fonte: Assembly for the SNES.

2.4. Códigos de Exemplo

- O código deve usar recuo quando houver rótulos, subrótulos ou rótulos de mais/menos na mesma linha de uma instrução. A quantidade de recuo é igual ao comprimento do tipo de etiqueta mais longo mencionado no bloco de código, incluindo dois pontos (":"), mais dois espaços adicionais.
    - O código deve usar espaços em branco para recuo, não tabulações.
- Opcodes são escritos inteiramente em maiúsculas (por exemplo: LDA).
- Os rótulos são escritos em PascalCase (por exemplo: Label1:).
- Os subrótulos são escritos inteiramente em minúsculas, sublinhado e o nome deve semanticamente se adequar ao parênteses, sem redundância (por exemplo: .return).
- Define são escritos em PascalCase (por exemplo: !SomeDefine).

- Dados diretos (db, dw, dl, dd), indexadores (,x, ,y, ,s) e especificadores de comprimento de opcode (.b, .w, .l) são escritos inteiramente em minúsculas.
- Os indicadores de comentários (ou seja ;) devem começar na coluna 20 e preenchidos à esquerda por espaços em branco, não por tabulações.
    - Se não houver espaço para um comentário na coluna 20, ele deve começar na mesma linha de qualquer maneira.
- Na verdade, nunca use abas.
- Os indicadores de comentário devem ser seguidos por um espaço em branco, antes do próprio comentário.
- Haverá uma nova linha extra após os opcodes RTS, RTL, RTI, JMP, JML, BRA, BRL.
- Estas são orientações que devem ser seguidas o mais rigorosamente possível, mas podem existir casos excepcionais. Use seu melhor julgamento.

Exemplo:

```
SomeLabel:        ;Este rótulo está em sua própria linha
LDA.b #$42
STA $00           ;Este é um comentário
RTS

.table:
db $01,$02,$03,$04 ;Outro comentário

.second_table:
db $01,$02,$03,$04,$01,$02,$03,$04 ;Um comentário excepcional



TestLabel:  LDA #$02 ;Esta etiqueta está na mesma linha de uma instrução
        STA $01
        BNE +
        NOP
```

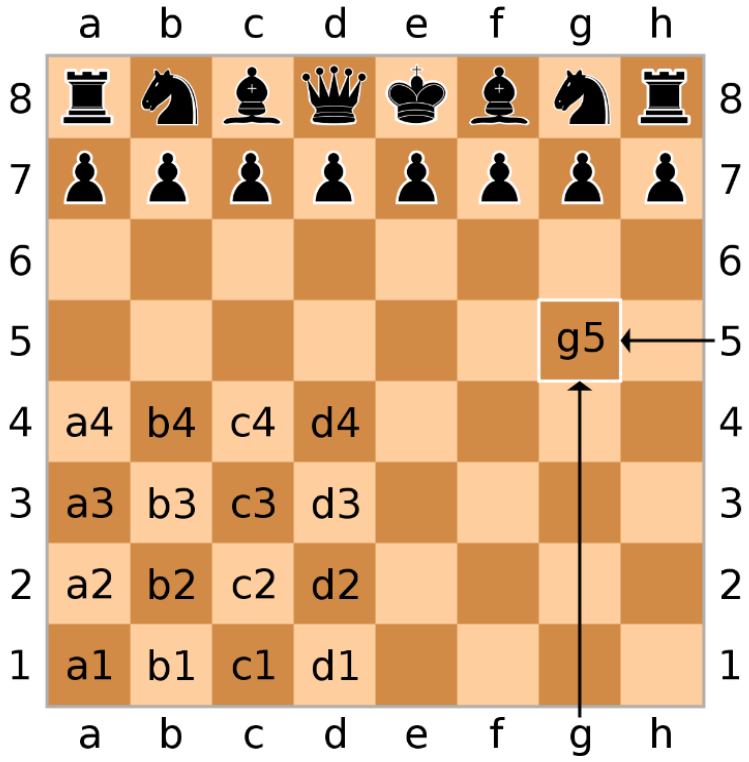
```
+       RTS     ;O código é recuado de acordo com "TestLabel", não "+"
```

2.5. A Memória do SNES

Escrever assembly envolve escrever um monte de instruções onde você carrega um "valor" e o armazena em um "endereço" para obter um efeito desejado, como alterar o power up do jogador. Ao escrever assembly, você trabalhará com a memória SNES na maior parte do tempo.

A memória do SNES é basicamente uma região de bytes, e cada byte está localizado em um "endereço". Pense nisso como um tabuleiro de xadrez:

Figura 3 - Exemplo de organização do endereçamento da memória.



Fonte: Assembly for the SNES.

Você pode ver que para se referir a uma determinada célula, a imagem faz uso de nomes de colunas e células. O "endereço" da rainha mostrada (o "valor") seria o endereço D8, por exemplo. Além disso, uma única célula não pode conter duas unidades. Este mesmo conceito se aplica à memória SNES.

A memória SNES é mapeada do endereço $000000 para $FFFFFF, embora apenas $000000-$7FFFFFF seja usado na maioria dos casos. O formato de um endereço é o seguinte: $BBHHDD.

- BB é o "byte bancário" do endereço
- HH é o "byte alto" do endereço
- DD é o "byte baixo" do endereço

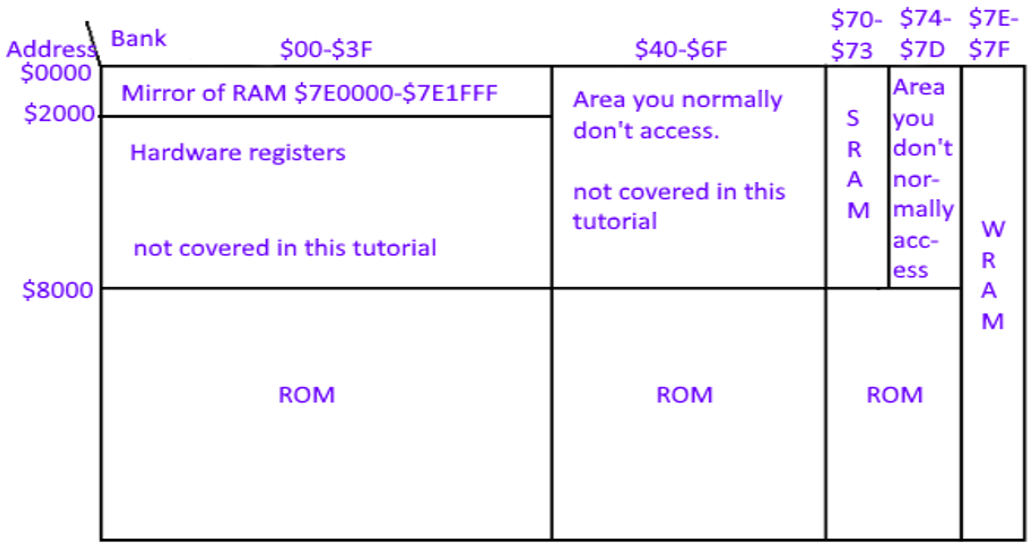
Os endereços podem ser escritos de 3 maneiras: $BBHHDD, $HHDD e $DD, como $7E0003, $0003 e $03.

- Endereços na notação $DD são chamados de "página direta"
- Endereços na notação $HHDD são chamados de "endereços absolutos"
- Endereços na notação $BBHHDD são chamados de "endereços longos"

Conforme estabelecido anteriormente, um endereço pode conter apenas um byte. Se você acessar um determinado endereço no modo de 16 bits, significa que você realmente acessa "endereço" e "endereço+1", porque um número de 16 bits consiste em dois bytes.

Aqui está um desenho para obter uma visão geral da memória básica do SNES (também conhecida como mapa de memória):

Figura 4 - O mapa de memória “LoROM”



Fonte: Assembly for the SNES.

ROM

ROM significa "Read-Only Memory" e é exatamente isso: memória que só pode ser lida. Isso significa que você não pode alterar a ROM armazenando valores nela com ASM. Você pode dizer que é o próprio jogo ou programa, que contém todo o código ASM e tabelas de dados, além de recursos como gráficos, música e assim por diante. Alternativamente: É o .smc/.sfc/.fig/etc. arquivo que você carrega em emuladores.

RAM

RAM significa "Memória de Acesso Aleatório". Esta é a memória dinâmica que permite que qualquer coisa seja gravada a qualquer momento. Você poderia dizer que este é o lugar onde você tem variáveis que são importantes e têm significado. A RAM pode ser gravada para obter um determinado efeito. Por exemplo, se você escrever $04 nas vidas extras do jogador, então o jogador terá exatamente 4 vidas extras.

A RAM do SNES tem 128kB de tamanho e está localizada nos endereços $7E0000-$7FFFFF. A RAM do SNES é completamente genérica. Não existe uma regra como "endereço $7E0120 é usado para vidas em todos os jogos SNES de todos os tempos". Você mesmo define o propósito da RAM, escrevendo o código ASM.

O mapa de memória mostra que os bancos $00-3F contém um "espelho" de RAM. Os endereços de RAM espelhados são endereços que contém o mesmo valor em todos os bancos. Isso significa que o endereço de RAM $001234 contém exatamente o mesmo valor que $0F1234 em todos os momentos. Ter a RAM espelhada significa que o código em execução na ROM nesses bancos pode acessar a RAM $7E0000 - $7E1FFF mais "facilmente". Por outro lado, o código executado em bancos $40-6F tem mais problemas para acessar a RAM porque a RAM não é espelhada lá.

Para simplificar, você sempre pode assumir que o banco $00 é igual ao banco $7E.

SRAM

SRAM significa "Memória de Acesso Aleatório Estático". Também é 128kB grande, e está localizado em blocos de 32kB em $ 700.000 - $ 707FFF, $ 710.000 - $ 717FFF, $ 720.000 - $ 727FFF e $ 730.000 - $ 737FFF, embora o tamanho final da SRAM dependa das próprias especificações da ROM, graças a algo chamado "cabeçalho ROM interno. O SRAM não é espelhado em outros bancos.

A SRAM se comporta exatamente como a RAM; você pode armazenar qualquer coisa e carregar qualquer coisa, mas os valores não são apagados quando o SNES é reiniciado. A SRAM é mantida viva com uma bateria de célula-botão real em um cartucho SNES real. Quando a bateria morre ou é removida, a SRAM não funcionará corretamente e possivelmente perderá dados após cada reinicialização. Nos emuladores, a SRAM é armazenada nos conhecidos arquivos ".srm".

A SRAM é geralmente usada para salvar arquivos, embora também possa ser usada como RAM extra.

3 Formato das instruções

3.1. Hexadecimal

Para programar no ASM 65c816, você precisará entender o básico do hexadecimal. Hexadecimal, também conhecido como "hex", é um sistema de contagem muito parecido com o decimal, que é o sistema de contagem diário que as pessoas usam. Em hexadecimal, há 6 dígitos adicionais por valor posicional, que são denotados através dos valores A-F, conforme a tabela abaixo.

Tabela 3 - Decimal/Hexadecimal

Decimal	Hexadecimal
0	0
1	1

2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11
...	...
255	FF

Fonte: Assembly for the SNES.

Existem várias maneiras de escrever números hexadecimais para não confundi-los com números decimais reais. Eles são os seguintes:

- Prefixo números hexadecimais com "0x" (por exemplo, 0x42)
- Prefixo números hexadecimais com "$" (por exemplo, $42)
- Sufixo números hexadecimais com "H" (por exemplo, 42H)

A convenção é prefixar números hexadecimais com "$".

Na montagem, um número hexadecimal com dois dígitos é chamado de "byte". Isso significa que valores entre $00-$FF são considerados um byte.

Valores Assinados e Não Assinados

No mundo real, os números podem ser positivos ou negativos. Em assembly, dependendo do código, os valores podem ser tratados como "assinados" ou "não assinados". Valores com sinal significam que eles também podem ser negativos: o valor $80 e superior são considerados números negativos em decimal, começando em -128, e em contagem regressiva à medida que o número hexadecimal está aumentando, como você pode ver na tabela abaixo.

Tabela 4 - Valores assinados e não assinados

Decimal	Hexadecimal
126	$7E
127	$7F
-128	$80
-127	$81
...	...
-1	$FF

Fonte: Assembly for the SNES.

A presença de números negativos depende da programação do jogo. Por exemplo, um jogador pode ter velocidade positiva e negativa (resultando em avançar ou retroceder), mas um jogador não pode ter vidas ou pontos extras negativos (porque normalmente isso não faz sentido). Escusado será dizer que o valor -0 não existe.

Valores Hexadecimais de Quatro Dígitos

Os números hexadecimais podem contar bem além de dois dígitos, como você pode ver abaixo.

Tabela 5 - Hexadecimal de quatro dígitos

Decimal	Hexadecimal
254	$FE
255	$FF
256	$0100
257	$0101
...	...
65535	$FFFF

Fonte: Assembly for the SNES.

O formato desse número hexadecimal é o seguinte: $HHLL.

- HH é o "byte alto" do número.
- LL é o "byte baixo" do número.

3.2. Binário

Outro sistema de contagem importante é o "binário". O binário tem apenas dois dígitos possíveis para cada valor posicional: 0 e 1. Um dígito binário também é chamado de "bit". Na sintaxe do assembly, os bits são prefixados por "%".

Um byte é composto por oito "bits". Como um dígito binário tem dois valores possíveis e um byte tem 8 bits, isso significa que existem 2^8 valores possíveis em um byte.

Por exemplo, um byte pode consistir nos seguintes bits: 1001 0110 ou 1001 0101. O primeiro bit da esquerda é chamado de "bit 7" e o bit final é chamado de "bit 0". Eles NÃO são chamados de bits 0-7, nem bits 8-1. Aqui está uma visão geral:

Outro sistema de contagem importante é o "binário". O binário tem apenas dois dígitos possíveis para cada valor posicional: 0 e 1. Um dígito binário também é chamado de "bit". Na sintaxe do assembly, os bits são prefixados por "%".

Um byte é composto por oito "bits". Como um dígito binário tem dois valores possíveis e um byte tem 8 bits, isso significa que existem 2^8 valores possíveis em um byte.

Por exemplo, um byte pode consistir nos seguintes bits: 1001 0110 ou 1001 0101. O primeiro bit da esquerda é chamado de "bit 7" e o bit final é chamado de "bit 0". Eles NÃO são chamados de bits 0-7, nem bits 8-1. Aqui está uma visão geral:

Tabela 6 - Bits

1	Bit 7654 3210
2	
3	1001 0110
4	1001 0101
5	....

Fonte: Assembly for the SNES.

A tabela abaixo mostra uma maneira relativamente fácil de memorizar binários, pois exibe um padrão.

Tabela 7 - Padrões Binário e Hexadecimal

Binário	Hexadecimal
%0000 0001	$01
%0000 0010	$02
%0000 0100	$04
%0000 1000	$08
%0001 0000	$10
%0010 0000	$20
%0100 0000	$40
%1000 0000	$80

Fonte: Assembly for the SNES.

Observe que há um espaço entre 4 bits para facilitar a leitura, embora os montadores geralmente não aceitem essa sintaxe. Grupos de 4 bits são chamados de "nibbles" e, para os propósitos deste capítulo, eles existem para tornar o binário mais fácil de ler, porque um nibble corresponde a um dígito em hexadecimal.

O SNES é capaz de trabalhar com números de 8 bits e 16 bits. Enquanto os números de 8 bits são chamados de byte, os números de 16 bits são chamados de "palavra". Eles parecem ter 16 bits em binário (por exemplo, 10000101 11010101, que é $855 em hexadecimal). No caso de números de 16 bits, o bit mais à esquerda é chamado de "bit 15", enquanto o bit mais à direita é chamado de "bit 0":

Observe que há um espaço entre 4 bits para facilitar a leitura, embora os montadores geralmente não aceitem essa sintaxe. Grupos de 4 bits são chamados de "nibbles" e, para os propósitos deste capítulo, eles existem para tornar o binário mais fácil de ler, porque um nibble corresponde a um dígito em hexadecimal.

O SNES é capaz de trabalhar com números de 8 bits e 16 bits. Enquanto os números de 8 bits são chamados de byte, os números de 16 bits são chamados de "palavra". Eles parecem ter 16 bits em binário (por exemplo, 10000101 11010101, que é $855 em hexadecimal). No caso de números de 16 bits, o bit mais à esquerda é chamado de "bit 15", enquanto o bit mais à direita é chamado de "bit 0":

Tabela 8 - Leitura de 16 bits

1	1111 11 (leia de cima para baixo)
2	Bit 5432 1098 7654 3210
3	1000 0101 1101 0101
4	0000 0000 1001 0110
5	**.... ....**

Fonte: Assembly for the SNES.

Tabela de descompactação dos dados do jogo Super Metroid do SNES de 1994, incluindo o binário, para o entendimento de como o binário se organiza para compor as instruções:

Tabela 9 - Rotina de decomposição do Super Metroid

Rotina de Decomposição [$80:B0FF-$80:B270]					
Nome do comando	Hex	Binário	Sintaxe	Alcance	Notas
Cópia direta	$00 …	%00000000 CMD.....	[%000xxxxx : xxxxx = #]	1-32 bytes	Copia os próximos (# + 1)

	$1F	%00011111			bytes dos Dados Compactados (CD) para a RAM.
Preenchimento de bytes	$20 … $3F	%00100000 CMD..... %00111111	[%001xxxxx : xxxxx = #]	1-32 bytes	Grava o próximo byte (# + 1) bytes na memória RAM.
Preenchimento de palavras	$40 … $5F	%01000000 CMD..... %01011111	[%010xxxxx : xxxxx = #]	1-32 bytes	Grava a próxima palavra (# + 1) bytes na memória RAM.
Preenchimento Sigma	$60 … $7F	%01100000 CMD..... %01100000	[%011xxxxx : xxxxx = #]	1-32 bytes	Grava o próximo byte na RAM. Em seguida, adiciona 1 a esse byte e o grava novamente... etc. Grava (# + 1) vezes.
Cópia da Biblioteca	$80 … $9F	%10000000 CMD..... %10011111	[%100xxxxx : xxxxx = #]	1-32 bytes	Cópia (# + 1) bytes do endereço de RAM fornecido nos próximos dois bytes.
Cópia EOR	$A0 … $BF	%10100000 CMD..... %10111111	[%101xxxxx : xxxxx = #]	1-32 bytes $7F:0000 - $7F:FFFF	Muito parecido com a cópia da biblioteca. A única diferença é que todos os dados copiados são EORed com %11111111.
Cópia Menos	$C0 … $DF	%11000000 CMD..... %10111111	[%110xxxxx : xxxxx = #]	1-32 bytes 0-255 do atual Y	Subtrai o próximo byte de Y, o deslocamento da RAM e cópia (# + 1) bytes para o Y atual. Pode copiar o último bloco.
CMD estendido	$E0 …	%111 000 00 | | | CMD—-	-->Bits 8 9 of X (aka #) ->Código de Comando.	1-1024 bytes $7F:0000	Estende o intervalo de CMDs anteriores de 1-32 a 1-1024 bytes de

	$FE	%111 111 11	Qualquer um dos 7 padrões CMD anteriores pode ir aqui.	- $7F:FFFF RAM	profundidade. Especificamente, os bits 7-5 sinalizam o ECMD e os bits 4-2 são interpretados como um CMD. Os bits 1-0 e o próximo byte são tratados como um valor de 10 bits para X, também conhecido como #.
Terminate SR	$FF	%11111111	[%11111111]	----------	Termina a sub-rotina

Fonte: Super Metroid Mod manual.

Cada linha é um instrução sequencial (opcodes e estrutura de dados) para a máquina virtual que é o decodificador.

O parâmetro (# == Número Símbolo). Neste caso, o número de bytes que são (devem ser) copiados do fluxo de entrada (dados compactados) para a saída (imagem na RAM).

Eles são descrição (opcodes e estrutura de dados) para a máquina virtual que é o decodificador.

Por exemplo:

Copia os próximos (# + 1) bytes dos Dados Compactados (CD) para a RAM.

É descrito em detalhes ao ler cada entrada.

- Nome do comando
    - Cópia direta

- Intervalo de codificação em hexadecimal
    - $00..$1F

- Intervalo de codificação em binário
    - %0000 0000 - %0001 1111

- Sintaxe

    - [%000x xxxx : x xxxx = #] significando um byte (% -> em binário) contendo 000 como (sua) porção de opcode, enquanto x xxxxi deve ser tomado como um valor (parâmetro) adicionalmente chamado # (aqui o comprimento).

- Faixa do Comando (Parâmetros)
    - 1..32

- Notas/Descrição
    - Copia os próximos (# + 1) bytes dos Dados Compactados (CD) para a RAM.

Os dados da sala, descrita na tabela, serão executados pela máquina, para transformar a entrada (seu programa) em saída (imagem na RAM).

- 01 -> Opcode 0 (Copiar dados) com comprimento de dados 2 (%00.0001 + 1)
    - 00 0A -> Dois bytes de dados a serem copiados para a saída.

- E9 3F -> Estendido (comprimento) Opcode 7 Sub-Opcode 2 (Write Word) com comprimento 320 (%01.0011.1111 + 1)
    - 44 80 -> Dois bytes de dados de palavra para serem escritos como 64 bytes para saída (32 vezes).

- 48 -> Opcode 2 (Write Word) com comprimento 9 (%00.1000 + 1)
    - 11 89 -> Dois bytes de dados de palavras para serem escritos como 9(!) bytes para saída (4 1/2 vezes).

... e assim por diante ...

Bandeiras

O binário é imensamente útil quando você está dando a um valor hexadecimal vários propósitos, como um botão liga/desliga para certos recursos. Esses bits são chamados de "flags" e geralmente são usados para economizar espaço na memória de trabalho dos jogos.

Por exemplo, você pode dividir um byte em 8 bits com cada bit tendo um significado diferente. Bit 7 pode indicar que um nível tem chuva ou não. O bit 6 pode

indicar que um layout de nível é horizontal ou vertical. O bit 5 pode indicar que a configuração de nível é durante o dia ou à noite, etc. Dessa forma, você pode compactar as informações em um único byte. Ficaria assim em binário:

Tabela 10 - Bandeiras

1	10100000
2	|| └──── Bandeira "é diurno"
3	| └──── Sinalizador "É nível horizontal"
4	└──── Bandeira "Está chovendo"

Fonte: Assembly for the SNES.

Finalmente, aqui está uma visão geral de como contar em decimal, hexadecimal e binário:

Tabela 11 - Visão geral de decimal, hexadecimal e binário

Decimal	Hexadecimal	Binário
00	$00	%0000 0000
01	$01	%0000 0001
02	$02	%0000 0010
03	$03	%0000 0011
04	$04	%0000 0100
05	$05	%0000 0101
06	$06	%0000 0110
07	$07	%0000 0111
08	$08	%0000 1000
09	$09	%0000 1001
10	$0A	%0000 1010
11	$0B	%0000 1011
12	$0C	%0000 1100

13	$0D	%0000 1101
14	$0E	%0000 1110
15	$0F	%0000 1111
16	$10	%0001 0000
17	$11	%0001 0001
...	...	...
254	$FE	%1111 1110
255	$FF	%1111 1111

Fonte: Assembly for the SNES.

Notação

Às vezes, os bits podem ser escritos de forma inconsistente, como 11 ou 110 0000. Isso torna o número binário mais difícil de ler, porque a convenção geral é escrever bits em grupos de oito. Para lê-los, você precisará adicionar 0s iniciais aos dígitos até que haja 8 bits ou 16 bits no total.

Em 8-bit:

- 11 se torna 00000011
- 1100000 torna-se 01100000

Em 16-bit:

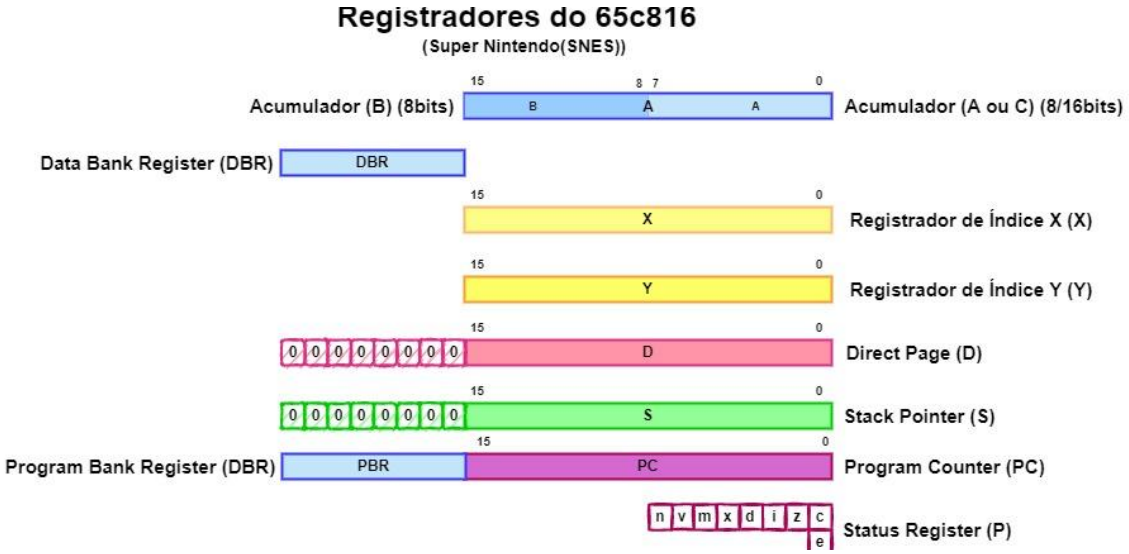
- 11 torna-se 00000000 00000011
- 1100000 torna-se 00000000 01100000

Você pode converter entre decimal, hexadecimal e binário, usando o modo de "programação" da calculadora do Windows. Há também muitas calculadoras online

que podem fazer isso. A sintaxe do assembly também aceita decimal e binário, então você normalmente não precisa converter entre decimal e hexadecimal.

4 CONJUNTO DE REGISTRADORES

Figura 5: Conjunto de registradores do SNES



Fonte: CARVALHO, D. M. (2022).

Como pode-se observar na imagem acima, o Super Nintendo possui a maioria de seus registradores em 16 bits, além de funcionar utilizando o conceito de banco de dados, que será tratado ao decorrer do relatório. Sendo assim, observe como desempenha cada um desses registradores:

Acumulador (A ou C):

O acumulador é o registrador de propósito geral. Quase todas as instruções aritméticas e lógicas usam o Acumulador, como, soma, shift, and, etc.

O acumulador tem uma arquitetura de 16 bits, porém utilizando o registrador nomeado Status Register, é possível especificar se o acumulador funcionará com 8 ou com 16 bits, isto posto, observa-se que ele pode ser tratado como 2 registradores de 8 bits (A e B), salientando ainda que quando tratado como um único registrador de 16 bits, geralmente utilizamos a letra C para representar o acumulador.

Registradores de índice X e Y (X e Y):

Os registradores X e Y possuem cada um, 16 bits e são usados na geração dos endereços efetivos e como contadores, ou seja, são usados para podermos formar o endereço final de onde vamos acessar os dados para realizar as instruções (Soma o A com algo que está na memória).

É importante destacar que apesar de parecidos, os registradores X e Y não são idênticos, tendo isto dito, é possível observar que alguns modos de endereçamento usam apenas um deles (X ou Y).

Direct Page (D):

Utiliza-se o registrador D para que possamos colocar este registrador apontando para um endereço e então todos os seus acessos ficam relativos a este endereço. por exemplo podemos guardar um valor 1000 bits dentro do Direct Page, então, quando utilizamos uma instrução de shift em 5 bits por exemplo, somaremos 1000+5, então fazer o shift com o valor 1005. O direct page tem 16 bits, assim como a maioria dos registradores presentes no SNES, e com pode-se observar na figura 5, os vários 0s à esquerda do nome do registrador D, significa que seu valor inicial é zero.

Stack Pointer (S):

O Stack Pointer possui 16 bits, e serve exclusivamente para salvar o endereço do topo da pilha (Stack) e assim como mencionado no registrador anterior, os vários 0s observados à esquerda do nome do registrador (figura 5), servem para dizer que seu valor inicial é zero.

Data Bank Register:

O Data Bank Register contém o banco padrão para transferências de memória.

Program Bank Register:

O Program Bank contém o endereço do banco de todas as buscas de instruções.

Program Counter (PC):

O Program Counter possui 16 bits + 8 bits do Program Bank Register, totalizando 24 bits (informações sobre os bancos de dados serão mencionados no decorrer deste texto).

O PC contém o endereço de memória que será utilizado para buscar a próxima instrução a ser executada pela CPU. Antes de executar qualquer instrução, a CPU envia o conteúdo de PC para a memória, através do Barramento de Endereço, a memória envia o conteúdo da memória nesse endereço, através do Barramento de Dados. Esse conteúdo é então armazenado no IR (instruction register), que no caso do SNES, é substituído pelo registrador de propósito geral.

É o registrador, então, que armazena o endereço de memória, do início da próxima instrução a ser executada. Após a leitura de um byte de uma instrução, o contador do programa é incrementado, apontando para o seu próximo byte (se houver).

Ao final da instrução, o contador do programa sempre armazena o endereço da próxima instrução a ser executada. O valor do contador do programa pode mudar de forma não sequencial quando alguma instrução de desvio ou chamada de sub-rotina é executada, sendo um novo endereço carregado neste registrador.

Status Register(P):

É o registrador que guarda os bits com os estados da cpu. Bits individuais são implícita ou explicitamente lidos e/ou escritos pelas instruções de código de máquina em execução no processador. O Status Register permite que uma instrução aja de acordo com o resultado de uma instrução anterior.

No Snes podemos ter os modos 8 bits e 16 bits, e esses modos são configurados por duas flags da Status Register. As flags ‘m’ e ‘x’ da Status Register são as que controlam os modos 8 e 16 bits. Desta forma a flag ‘m’ controla se o registrador A trabalha em modo 8 ou 16 bits. Se a flag estiver em 0 o modo é 16 bits. Se estiver em 1 o modo é 8 bits. E a flag x controla se os registradores X e Y trabalham em modo 8 ou 16 bits. Se a flag estiver em 0 o modo é 16 bits. Se estiver em 1 o modo é 8 bits.

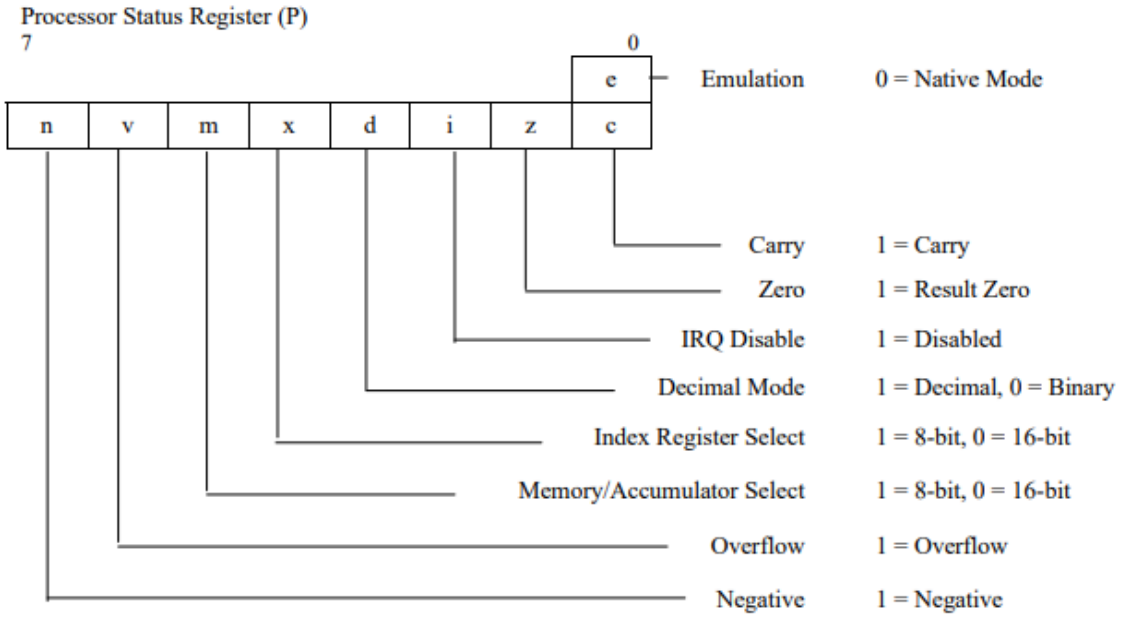
Essas flags alteram o tamanho dos registradores e também todas as instruções que trabalham com esses registradores. Então se o A estiver em modo 8 bits, ao ler um dado da memória com a instrução lda, a Cpu lê apenas um byte. Se estiver em modo 16 bits a CPU lerá 2 bytes. Este é só um exemplo, mas o mesmo vale para todas as instruções que usam esses registradores.

Tabela 12: Flags armazenadas no Stack Pointer (P)

Mnemônica	Valor	Valor Binário	Descrição
N	#$80	10000000	Negativo
V	#$40	01000000	Overflow
M	#$20	00100000	Tamanho do registrador Acumulador (apenas para o modo nativo), (0 = 16-bit, 1 = 8-bit)
X	#$10	00010000	Tamanho do registrador de índice (apenas para o modo nativo), (0 = 16-bit, 1 = 8-bit)
D	#$08	00001000	Modo decimal
I	#$04	00000100	Desabilita o IRQ
Z	#$02	00000010	Zero
C	#$01	00000001	Carry
E			(6052) Modo de Emulação
B	#$10	00010000	Instrução Break (apenas para o modo de emulação)

Fonte: CARVALHO, D. M., 2022.

Figura 6: Processos do Status Register (P)



Fonte: wiki.superfamicom.org.

Figura 7, 8 e 9 - Tabela com operadores, código das operações e status dos registros, contendo os opcodes dos registradores:

Mnemonic	Operation	Opcodes																						Status Register							
																								7	6	5	4	3	2	1	0
	Addressing Mode	a	A	a,x	a,y	al	al,x	(a)	(a,x)	d	d,s	d,x	d,y	(d)	[d]	(d,s),y	(d,x)	(d),y	[d],y	i	r	rl	s	xyc	#	N	V	M	X	D	I	Z	C
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	N	V	1	B	D	I	Z	C
ADC	A+M+C→A	6D		7D	79	6F	7F			65	63	75		72	67	73	61	71	77						69	N	V	.	.	.	.	Z	C
AND	A^M→A	2D		3E	39	2F	3F			25	23	35		32	27	33	21	31	37						29	N	.	.	.	.	.	Z	.
ASL	C←15/7 6 ... 10 ←0	0E	0A	1E						06		16														N	.	.	.	.	.	Z	C
BCC	Branch if C = 0																				90					.	.	.	.	.	.	.	.
BCS	Branch if C = 1																				B0					.	.	.	.	.	.	.	.
BEQ	Branch if Z = 1																				F0					.	.	.	.	.	.	.	.
BIT	A ^ M (Note 1)	2C		3C						24		34													89	M	M	.	.	.	.	Z	.
BMI	Branch if N = 0																				30					.	.	.	.	.	.	.	.
BNE	Branch if Z = 0																				D0					.	.	.	.	.	.	.	.
BPL	Branch if N = 0																				10					.	.	.	.	.	.	.	.
BRA	Branch Always																				80					.	.	.	.	.	.	.	.
BRK	Break (Note 2)																						00			.	.	.	•	0	1	.	.
BRL*	Branch Long Always																					82				.	.	.	.	.	.	.	.
BVC	Branch if V = 0																				50					.	.	.	.	.	.	.	.
BVS	Branch if V = 1																				70					.	.	.	.	.	.	.	.
CLC	C → 0																			18						.	.	.	.	.	.	.	0
CLD	0 → D																			D8						.	.	.	.	0	.	.	.
CLI	0 → 1																			58						.	.	.	.	.	0	.	.
CLV	0 → V																			B8						.	0	.	.	.	.	.	.
CMP	A-M	CD		DD	D9	CF	DF			C5	C3	D5		D2	C7	D3	C1	D1	D7						C9	N	.	.	.	.	.	Z	C
COP*	Co-Processor																						02			.	.	.	.	0	1	.	.
CPX	X-M	EC								E4															E0	N	.	.	.	.	.	Z	C
CPY	Y-M	CC								C4															C0	N	.	.	.	.	.	Z	C
DEC	Decrement	CE	3A	DE						C6		D6														N	.	.	.	.	.	Z	.
DEX	X-1 → A																			CA						N	.	.	.	.	.	Z	.
DEY	Y-1 → Y																			88						N	.	.	.	.	.	Z	.
EOR	A xv M → A	4D		5D	59	4F	5F		5D	45	43	55		52	47	53	41	51	57						49	N	.	.	.	.	.	Z	.
INC	Increments	EE	1A	FE						E6		F6														N	.	.	.	.	.	Z	.
INX	X+1 → X																			E8						N	.	.	.	.	.	Z	.
INY	Y+1 → Y																			C8						N	.	.	.	.	.	Z	.
JML*	Jump Long to new location							DC																		.	.	.	.	.	.	.	.
JMP	Jump to new location	4C				5C		6C	7C																	.	.	.	.	.	.	.	.
JSL*	Jump long to subroutine					22																				.	.	.	.	.	.	.	.
JSR	Jump to Subroutine	20							FC																	.	.	.	.	.	.	.	.
LDA	M → A	AD		BD	B9	AF	BF			A5	A3	B5		B2	A7	B3	A1	B1	B7						A9	N	.	.	.	.	.	Z	.

Fonte: The Western Design Center Inc. (2018).

Continuação:

Mnemonic	Operation	OpCodes																						Status Register							
																								7	6	5	4	3	2	1	0
	Addressing Mode	a	A	a,x	a,y	al	al,x	(a)	(a,x)	d	d,s	d,x	d,y	(d)	[d]	(d,s),y	(d,x)	(d),y	[d],y	i	r	rl	s	xyc	#	N	V	M	X	D	I	Z	C
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	N	V	1	B	D	I	Z	C
LDX	M → X	AE			BE					A6			B6												A2	N	.	.	.	.	.	Z	.
LDY	M → Y	AC		BC						A4		B4													A0	N	.	.	.	.	.	Z	.
LSR	0 → 15/7 6 ... 1 0 → C	4E	4A	5E						46		56														0	.	.	.	.	.	Z	C
MVN*	M→M NEGATIVE																							54		.	.	.	.	.	.	.	.
MVP*	M→M POSITIVE																							44		.	.	.	.	.	.	.	.
NOP	No Operation																			EA						.	.	.	.	.	.	.	.
ORA	A V M → A	0D		1D	19	0F	1F			05	03	15		12	07	13	01	11	17						09	N	.	.	.	.	.	Z	.
PEA*	Mpc+1, Mpc+2 → Ms-1, Ms S-2 → S																						F4			.	.	.	.	.	.	.	.
PEI*	M(d), M(d+1) → Ms-1, Ms S-2 → S																						D4			.	.	.	.	.	.	.	.
PER*	Mpc+d, Mpc+d+1 → Ms-1, Ms S-2 → S																						62			.	.	.	.	.	.	.	.
PHA	A → Ms, S-1 → S																						48			.	.	.	.	.	.	.	.
PHB*	DBR → Ms, S-1 → S																						8B			.	.	.	.	.	.	.	.
PHD*	D → Ms, Ms-1, S-2 → S																						0B			.	.	.	.	.	.	.	.
PHK*	PBR → Ms, S-1 → S																						4B			.	.	.	.	.	.	.	.
PHP	P → Ms, S-1 → S																						08			.	.	.	.	.	.	.	.
PHX	X → Ms, S-1 → S																						DA			.	.	.	.	.	.	.	.
PHY	Y → Ms, S-1 → S																						5A			.	.	.	.	.	.	.	.
PLA	S + 1 → S, Ms → A																						68			N	.	.	.	.	.	Z	.
PLB•	S + 1 → S, Ms → DBR																						AB			N	.	.	.	.	.	Z	.
PLD•	S + 2 → S, Ms – 1, Ms → D																						2B			N	.	.	.	.	.	Z	.
PLP	S + 1 → S, Ms → P																						28			N	V	M	X	D	1	Z	C
PLX	S + 1 → S, Ms → X																						FA			N	.	.	.	.	.	Z	.
PLY	S + 1 → S, Ms → Y																						7A			N	.	.	.	.	.	Z	
REP•	M^P → P																								C2	N	V	M	X	D	1	Z	C
ROL	C ← 15/7 6 5 4 3 2 1 0 ← C	2E	2A	3E						26		36														N	.	.	.	.	.	Z	C
ROR	C →15/7 6 5 4 3 2 1 0 → C	6E	6A	7E						66		76														N	.	.	.	.	.	Z	C
RTI	Return from Interrupt																						40			N	V	M	X	D	1	Z	C
RTL*	Return from Sub. Long																						6B			.	.	.	.	.	.	.	.
RTS	Return from Subroutine																						60			.	.	.	.	.	.	.	.
SBC	A – M – (C) → A	ED		FD	F9		FF			E5	E3	F5		F2	E7	F3	E1	F1	F7						E9	N	V	.	.	.	.	Z	C
SEC	1 → C																			38						.	.	.	.	.	.	.	1
SED	1 → D																			F8						.	.	.	.	1	.	.	.
SEI	1 → 1																			78						.	.	.	.	.	1	.	.
SEP*	MVP → P																								E2	N	V	M	X	D	1	Z	C
STA	A → M	8D		9D		8F	9F			85	83	95		92	87	93	81	91	97							.	.	.	.	.	.	.	.
STP	STOP (1 → PH12)																			D8						.	.	.	.	.	.	.	.
STX	X → M	8E								86			96													.	.	.	.	.	.	.	.

Fonte: The Western Design Center Inc. (2018).

Continuação:

Mnemonic	Operation	OpCodes																							Status Register							
																									7	6	5	4	3	2	1	0
	Addressing Mode	a	A	a,x	a,y	ai	ai,x	(a)	(a,x)	d	d,s	d,x	d,y	(d)	[d]	(d,s),y	(d,x)	(d),y	[d],y	i	r	rl	s	xyc	#	N	V	M	X	D	I	Z	C
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	N	V	1	B	D	I	Z	C
STY	Y → M	8C		9E						84		94																					
STZ	00 → M	9C		9E						64		74																					
TAX	A → X																			AA						N						Z	
TAY	A → Y																			AB						N						Z	
TCD*	C → D																			5B						N						Z	
TCS*	C → S																			1B													
TDC*	D → C																			7B						N						Z	
TRB		1C								14																						Z	
TSB	AVM → M	0C								04																						Z	
TSC*	S → C		3B																							N						Z	
TSX	S → X		BA																							N						Z	
TXA	X → A		8A																							N						Z	
TXS	X → S		9A																														
TXY*	X → Y		9B																							N						Z	
TYA	Y → A		98																							N						Z	
TYX*	Y → X		BB																							N						Z	
WAI	0 → RDY		CB																														
WDM*	No Operation		42																														
XBA*	B → A		EB																							N						Z	
XCE*	M → Y		FB																														E

Fonte: The Western Design Center Inc. (2018).

5 Modo de endereçamento

Existem diferentes modos de endereçamento no 65c816. Os modos de endereçamento são usados para fazer com que os opcodes acessem endereços e valores de maneira diferente, como "indexado" ou "direto indireto" (explicado posteriormente neste tutorial). Usando-os com sabedoria, você pode acessar valores e endereços de memória de várias maneiras. Por exemplo, você pode carregar imediatamente um valor em um registrador, como A, ou carregar um byte da ROM em A. Tenha em mente que nem todos os opcodes suportam todos os tipos de

modos de endereçamento. Aqui estão alguns dos modos de endereçamento importantes que você usará com muita frequência.

Imediato 8/16 bits

Este modo de endereçamento define um valor absoluto, que é escrito como #$XX no modo de 8 bits ou #$XXXX no modo de 16 bits. O # significa "valor imediato", enquanto o $ significa hexadecimal. Usar # sozinho torna a entrada decimal. Por exemplo, #10 é o mesmo que #$0A. Pense em um valor imediato como um número que você está definindo diretamente.

Página direta

Este modo de endereçamento define um endereço de página direto, que é escrito como $XX.

A página direta são os últimos 2 dígitos hexadecimais de um endereço longo. Por exemplo, o endereço $7E0011 como página direta seria $11. Ao carregar de um endereço de página direto, o byte do banco é SEMPRE tratado como $00, sem exceções. Se você escrever "LDA $ 11", por exemplo, você carregaria o conteúdo de $000011 no acumulador, que também é espelhado em $ 7E0011 (lembre-se da ilustração da memória do SNES). Portanto, você carrega o conteúdo de $ 7E0011 em A.

Absoluto

Este modo de endereçamento define um endereço absoluto, que é escrito como $XXXX.

Um endereço absoluto são os últimos 4 dígitos hexadecimais de um endereço longo. Usando o exemplo anterior, o endereço $7E0011 como endereço absoluto

seria $0011. O byte do banco do endereço absoluto é determinado pelo registrador do banco de dados.

Longo

Este modo de endereçamento define um endereço longo, que é escrito como $XXXXXX.

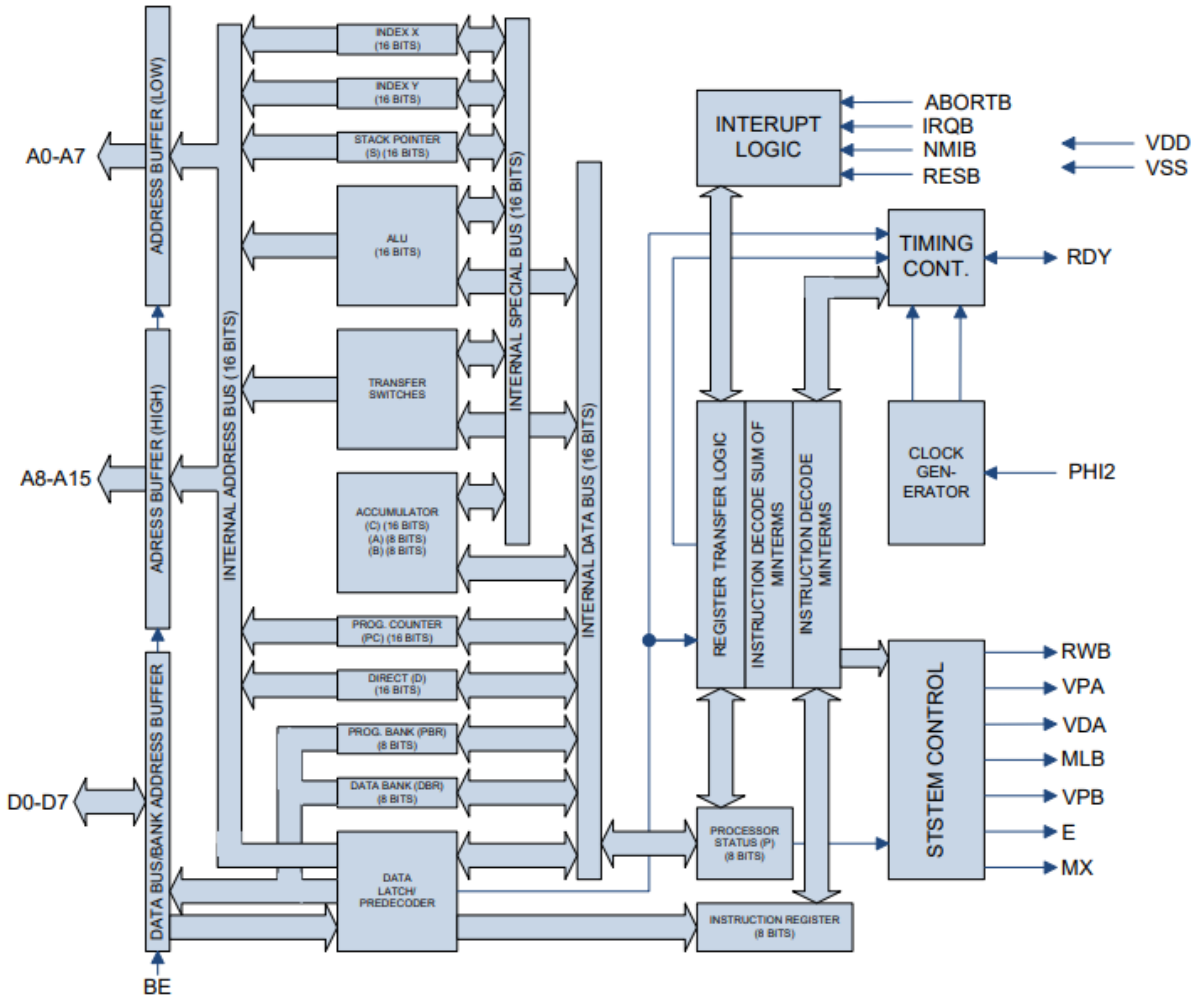
Endereços longos trazem menos complicações ao lidar com bancos e espelhamento. Você também não precisa se preocupar com o que o registro do banco de dados contém atualmente. Com endereços longos, você pode acessar qualquer endereço na memória do SNES.

Outros Modos de Endereçamento

O SNES suporta mais modos de endereçamento. Os modos de endereçamento acima são os básicos. Existem também modos de endereçamento, como: versões indexadas de página direta, endereços absolutos e longos e muito mais. Eles serão explicados no final deste tutorial porque você não precisa deles neste momento. Isso tornaria as coisas apenas mais confusas no momento.

6 DATA PATH

Figura 10 - Arquitetura Interna Diagrama de Blocos Simplificado do processador W65C816S.



Fonte: The Western Design Center Inc. (2018).

Figura 11 - Descrição da Função do Pino

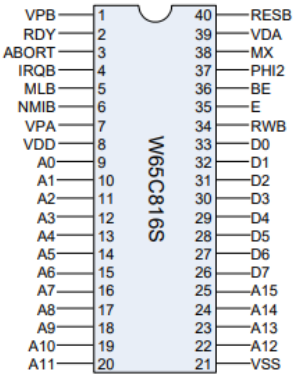


Figure 2-2 W65C816S 40 Pin DIP Pinout

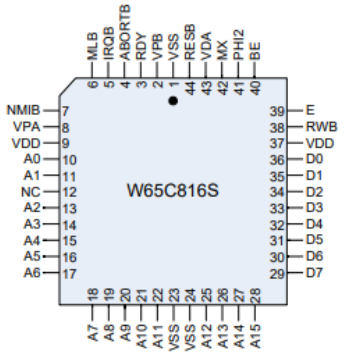


Figure 2-3 W65C816S 44 Pin PLCC Pinout

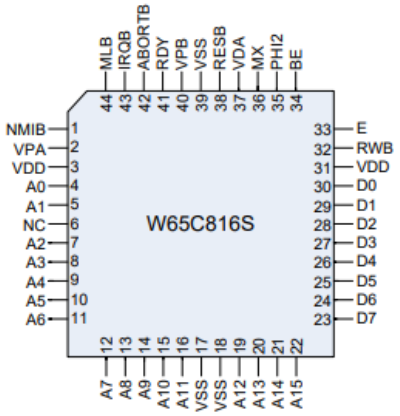


Figure 2-4 W65C816S 44 PIN QFP Pinout

Fonte: The Western Design Center Inc. (2018).

Tabela das Funções dos Pinos

Pin	Description
A0-A15	Address Bus
ABORTB	Abort Input
BE	Bus Enable
PHI2	Phase 2 In Clock
D0-D7	Data Bus/Bank Address Bus
E	Emulation OR Native Mode Select
IRQB	Interrupt Request
MLB	Memory Lock
MX	Memory and Index Register Mode Select
NC	No Connect
NMIB	Non-Maskable Interrupt
RDY	Ready
RESB	Reset
RWB	Read/Write
VDA	Valid Data Address
VPB	Vector Pull
VPA	Valid Program Address
VDD	Positive Power Supply
VSS	Internal Logic Ground

Fonte: The Western Design Center Inc. (2018).

7 Sintaxe introdutória do assembly (Asar)

Antes de introduzir os leitores aos exemplos em código, é necessário evidenciar que pode ocorrer discrepância em algumas palavras em relação a outras fontes, devido a tradução ou interpretação, como, por exemplo, labels que neste trabalho serão muitas vezes tratadas como sendo rótulos. Observa-se também que as seguintes informações podem igualmente ser encontradas nas documentações sobre a Asar, (Asar é um assembler para o Super Nintendo, baseado no xkas v0.06). Porém também serão descritas aqui para melhor entendimento do conteúdo abordado neste documento.

Defines (definir/definições):

Defines são basicamente definições de variáveis, de lógica muito parecida com algumas linguagem de alto nível como c e c++. Segue os exemplos a seguir:

Exemplo que define um valor imediato:

```
    !Value = $03


    LDA #!Value
    STA $01
```

Exemplo que define um endereço:

```
    !Address = $01


    LDA #$03
    STA !Address
```

Esteja ciente de que o Asar realmente faz uma pesquisa e substituição de texto simples, em vez de avaliar a expressão em uma definição. Em outras palavras, Asar não é inteligente o suficiente para descobrir que uma definição é um "endereço", "valor imediato" ou qualquer outra coisa. Aqui está um exemplo de uso impróprio de definição:

```
    !Value = #$03       ; Note o #


    LDA  #!Value            ; Uma pesquisa e substituição (search and
replace) transforma isso em  "LDA ##$03"
    STA $01             ; Portanto, ele irá resultar em um erro!
```

Labels (rótulos):

O processador SNES pode fazer uso de rótulos para determinar os locais para onde pode saltar. Os rótulos usados pelos opcodes são substituídos por valores reais que denotam os locais para os quais pular, endereços relativos ou absolutos.

Sub Labels (subrótulos):

Subrótulos são tipos especiais de rótulos que têm um parênteses e são prefixados com um ponto ("."), e não são sufixados com dois pontos (":"). Os subrótulos são úteis se você costuma usar rótulos que não são exclusivos (por exemplo, "retorno"). Aqui está um exemplo de um subrótulo "retorno" recorrente:

```
JSR Main
JSR Sub
RTS

Main:
LDA $10
BEQ .return
STA $11
.return
RTS

Sub:
LDA $20
BEQ .return
STA $21
.return
RTS
```

Observe que sub-rótulos não têm nenhuma regra em termos de estilo de escrita. Você pode capitalizar ou manter tudo em minúsculas. Neste exemplo, está tudo em minúsculas.

Relative labels (rótulos relativos):

Rótulos relativos são uma solução alternativa para subrótulos e geralmente são usados quando o código já é auto documentado o suficiente, por exemplo, quando o código precisa pular um único store dependendo de uma ramificação. Isso evita que você pense em um nome de rótulo, como "skipstorewhenplayerisbig". Rótulos relativos são escritos usando + e -. O positivo está à frente da instrução de desvio, enquanto o menos está atrás da instrução de desvio. Eles podem ser repetidos quantas vezes forem necessárias para denotar diferentes níveis de profundidade.

Exemplo de rótulos relativos:

```
LDA $10
BEQ +
STA $11
+
RTS
```

Este código pula um único store para $11 quando o endereço $10 tem o valor $00.

Aqui está outro exemplo, demonstrando uma ramificação para trás, causando um loop infinito:

```
- BRA -
```

Aqui está outro exemplo, demonstrando diferentes níveis de profundidade relativa do rótulo:

```
LDA $10
BEQ ++
LDA $11
BEQ +
```

```
STZ $12
+
STZ $13
++
STZ $14
RTS
```

●Se o endereço $7E0010 tiver o valor $00, o endereço $7E0014 será apagado.

●Se o endereço $7E0011 tiver o valor $00, os endereços $7E0013 e $7E0014 serão apagados

●Caso contrário, os endereços $ 7E0012, $ 7E0013 e $ 7E0014 estão todos liberados.

É possível escrever programas completamente desprovidos de valores e endereços fixos (também conhecidos como "números mágicos"), fazendo uso inteligente de rótulos e definições fora de ramificações e jumps(saltos). Quando você usa rótulos com instruções de carregamento, por exemplo, ele pega o endereço do rótulo e o usa como parâmetro. No entanto, você também pode usar rótulos como valores, em vez de endereços. Isso é especialmente útil ao configurar ponteiros indiretos, e é por isso que é importante poder pegar certas partes de um endereço em vez do endereço completo.

No exemplo a seguir, um LDA carregando o endereço de um rótulo como um valor ficaria assim:

```
LDA #somelabel
STA $00
```

Isso é problemático, porque o rótulo é montado em um valor de 24 bits, que é o endereço, e não há LDA que aceite um valor de 24 bits. Em vez disso, o LDA tenta pegar o maior valor suportado possível, assim pega o byte alto (high byte) e baixo (low byte) do valor (porque é de 16 bits). Mas se você estiver escrevendo código de

8 bits naquele momento, o código não será executado conforme o esperado e falhará.

Especificações de comprimento do Opcode:

Para ler um valor em um comprimento fixo e bem definido, você pode usar "especificadores de comprimento de opcode".

Figura 13: Notações especiais anexadas aos opcodes

Syntax	Definition	Description
.b	byte (8-bit)	Forces the parameter to be 8-bit
.w	word (16-bit)	Forces the parameter to be 16-bit
.l	long (24-bit)	Forces the parameter to be 24-bit

Fonte: Assembly for the SNES.

No exemplo anterior, você pode forçar o montador a usar apenas os bytes baixos do rótulo, usando .b:

```
LDA.b $somelabel
STA $00
```

O mesmo se aplica às definições. Você pode usá-las como valores e, usando uma especificação de comprimento de opcode, você pode pegar apenas algumas partes das definições em vez do valor completo. Por exemplo:

```
!Size = $7FFF


LDA #!Size
STA $00
```

Isso executaria como:

```
LDA #$7FFF
LDA #$7FFF
STA $00
```

Isso seria problemático no modo de 8 bits, pois isso é montado no modo de 16 bits. Para corrigir esse problema, você pode usar .b:

```
!Size - $7FFF
LDA.b #!Size
STA $00
```

Isso executaria como:

```
LDA #$FF
STA $00
```

Bitshifts (deslocamento de bits):

Expandindo o exemplo anterior:

```
!Size = $7FFF
LDA.b #!Size
STA $00
```

Se você quisesse armazenar o byte alto dessa definição no endereço $7E0001, em vez do byte baixo, você precisaria de uma maneira de pegar apenas o byte alto da definição. Para fazer isso, você terá que usar bitshifts:

Figura 14: Deslocamento de bits (Asar)

Syntax	Definition	Description
>>	Shift right	Shifts bits right n times
<<	Shift left	Shifts bits left n times

Fonte: Assembly for the SNES.

Lembre-se de que um byte consiste em 8 bits, portanto, você precisa "pular" 8 bits para pegar os próximos 8 bits que precisamos. Deslocando o bit 8 vezes para a direita, você descarta o byte inferior do valor:

```
!Size = $7FFF
LDA.b #!Size>>8    ; apenas $7F permanece
STA $01
```

Bitshifts são incrivelmente valiosos ao “pegar’ certas partes de endereços ou valores. Eles também podem ser usados em rótulos e, assim, você também pode pegar bytes do banco:

```
LDA.b #somelabel>>16 ; "pegar" o byte do banco de uma label(rótulo)
STA $01
```

O mesmo funciona para as definições (defines):

```
!Address = $7E8000


LDA.b #!Address>>16 ; Pegue $7E das definições (define)
STA $02
```

Construindo endereços:

Fazendo uso de bitshifts e especificadores de comprimento de opcode, é possível fornecer endereços para certas sub-rotinas, ou como endereços indiretos. Aqui está um exemplo que constrói um endereço indireto:

```
LDA.b #Sometable
STA $00
LDA.b #Sometable>>8
STA $01
LDA.b #Sometable>>16
STA $02
LDA [$00]          ; Isso carrega o valor $01 em A
RTS


Sometable: db $01,$02,$04,$08
```

Table sizes:

Existem situações em que é útil saber o tamanho das “tables” (para este projeto considere tables como sendo uma tabela), como para movimentos. Para obter o tamanho de uma tabela, você coloca um rótulo no início e no final de uma tabela, como este:

```
Sometable: db $01,$02,$04,$08
.end
```

Então, usando o operador de subtração, "-", você subtrai o endereço inicial e final da tabela, obtendo efetivamente o tamanho da tabela. Por exemplo:

```
LDA.b #Sometable_end-Sometable ; #$04
STA $00
RTS


Sometable: db $01,$02,$04,$08
.end
```

Observe que é importante usar especificadores de comprimento de opcode, pois ainda estamos lidando com rótulos, portanto, valores de 24 bits.

8 Conclusões

Durante a pesquisa e revisão sobre as informações a respeito do Super Nintendo Entertainment System, percebeu-se que apesar do SNES ser um dispositivo antigo (1999~1997), há de fato muita informação a ser reunida sobre o console. Desta forma apenas as informações mais essenciais para este projeto foram reunidas e dispostas neste documento, salvo, as informações não encontradas ou restritas a sua desenvolvedora (Nintendo).

Ao final do trabalho, os autores concluem que os objetivos iniciais foram em sua maior parte alcançados, deixando a desejar nos aspectos: Apresentar as unidades de syscall quando aplicável, Apresentar as unidade de tratamento de interrupção e Especificar e discutir a motivação de ser cisc ou risc. Porém espera-se ser possível (com as informações aqui dispostas) a continuação do trabalho em semestres seguintes, assim como previsto e objetivado ao início do projeto.

REFERÊNCIAS

ASSEMBLY FOR THE SNES. **Tutorial de programação com o uso da linguagem Assembly 65c816 usando o processador de arquitetura chip Ricoh 5A22.** Disponível em: https://ersanio.gitbook.io/assembly-for-the-snes/. Acesso em: 22 jun. 2022.

THE WESTERN DESIGN CENTER INC. **Documentação do Microprocessador W65C816S de 8/16-bit.** Disponível em: https://www.westerndesigncenter.com/wdc/documentation/w65c816s.pdf. Acesso em: 22 jun. 2022.

SUPER METROID MOD MANUAL. **Documentação para construção de uma réplica do jogo Super Metroid.** Disponível em: https://metroidconstruction.com/SMMM/#part-2. Acesso em: 22 jun. 2022.

EYES, David; LICHTY, Ron. **Programming the 65816: Including the 6502, 65C02 and 65802.** The Western Design Center, Inc., April/1992. Disponível em: https://wiki.superfamicom.org/uploads/assembly-programming-manual-for-w65c816.pdf. Acesso em: 21 jun. 2022.

RPGHACKER.GITHUB.IO. **A multi-architecture SNES assembler by Alcaro, modelled after xkas v0.06 by byuu.** Disponível em: https://rpghacker.github.io/asar/manual/#standard-includes. Acesso em: 23 jun. 2022.

SUPER FAMICOM DEVELOPMENT WIKI. **65816 Reference**. Disponível em: https://wiki.superfamicom.org/65816-reference. Acesso em: 23 jun. 2022.