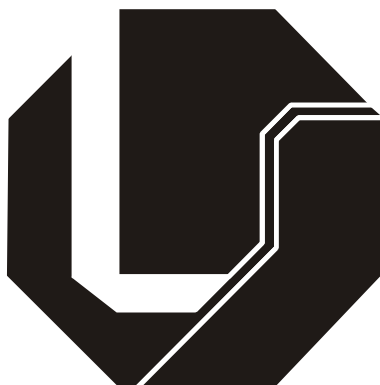


UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



**DESENVOLVIMENTO DE
FERRAMENTAS PARA
MANIPULAÇÃO E GERAÇÃO DE
ARQUIVOS MIDI-SMF
EM LINGUAGEM FUNCIONAL
CLEAN**

Hélcio Camargo Júnior

Abril
2007

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**DESENVOLVIMENTO DE
FERRAMENTAS PARA
MANIPULAÇÃO E GERAÇÃO DE
ARQUIVOS MIDI-SMF
EM LINGUAGEM FUNCIONAL
CLEAN**

Hélcio Camargo Júnior*

Texto da dissertação apresentada à
Universidade Federal de Uberlândia,
perante a banca de examinadores abaixo,
como parte dos requisitos necessários para
a obtenção do título de Mestre em
Ciências.

Banca Examinadora:

Prof. Luciano Vieira Lima, DSc. - Orientador (UFU)

Prof. Alcimar Soares Barbosa, DSc. (UFU)

Prof. Wellesley Barros Ferreira, DSc. (EAFUDI)

* A bolsa de estudo para esta pesquisa foi concedida pela CNPQ, Brasil.

FICHA CATALOGRÁFICA

Dados Internacionais de Catalogação na Publicação (CIP)

C172d Camargo Junior, Hécio.

Desenvolvimento de ferramentas para
manipulação e geração de arquivos MIDI-SMF em
linguagem funcional clean / Hécio Camargo Junior.
- 2007.

341 f. : il.

Orientador: Luciano Vieira Lima.

Dissertação (mestrado) – Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica.

Inclui bibliografia.

1. Engenharia elétrica - Teses. 2. Música por computador - Teses.
I. Lima, Luciano Vieira. II. Universidade Federal de Uberlândia.
Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU: 621.3

Elaborado pelo Sistema de Bibliotecas da UFU / Setor de Catalogação e Classificação

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**DESENVOLVIMENTO DE
FERRAMENTAS PARA
MANIPULAÇÃO E GERAÇÃO DE
ARQUIVOS MIDI-SMF
EM LINGUAGEM FUNCIONAL
CLEAN**

Hélcio Camargo Júnior

Dissertação apresentada por Hélcio Camargo Júnior à Universidade Federal de Uberlândia, como parte dos requisitos necessários para a obtenção do título de Mestre em Ciências.

Prof. Luciano Vieira Lima, DSc
Orientador

Prof. Darizon Alves de Andrade, Ph. D.
Coordenador do Curso de Pós-Graduação

Uxori Amantissimae

Agradecimentos

Primeiramente agradeço a Deus, que tem me dado saúde e vontade de querer trabalhar por esta terra.

Agradeço imensamente ao meu orientador Eng. Prof. Dr. Luciano Vieira Lima pelo incentivo, amizade, empenho, dedicação, e profissionalismo. Graças a sua valiosa orientação, pude adquirir vastos conhecimentos.

Agradeço a Sandra Lima pelas idéias e sugestões que contribuíram para o engrandecimento deste trabalho.

Agradeço a meu pai Hércio, que apesar de não se encontrar mais entre nós, tenho certeza que de alguma forma sempre me envia energias positivas. A minha mãe Gracinda, a minha irmã Helcione e ao meu avô Juca, por todo o amor, carinho, compreensão e respeito. Aos meus filhos Helvécio e Christiano que sempre me apoiaram e me incentivaram em meus propósitos.

Agradeço também ao Reny Cury Filho, Carlos Menezes e Júnia Magalhães, pelo companheirismo e por várias colaborações prestadas ao desenvolvimento deste trabalho.

Agradeço em especial ao meu grande amor, Maria Augusta, pelo companheirismo, compreensão, paciência, apoio e incentivo, e que bravamente suportou e soube entender os diversos momentos ruins que tivemos durante a realização deste trabalho.

Enfim, agradeço a todos os meus amigos que direta ou indiretamente contribuíram para a concretização deste sonho, e souberam entender a falta de tempo, o afastamento temporário que tivemos durante este processo.

Resumo

Este trabalho tem como uns dos objetivos principais produzir um material didático que permita ao leitor, programador ou músico, entender e se capacitar a editar e promover análises em arquivos MIDI SMF formato 1 e formato 0 em baixo nível. Tal capacitação se dá através do entendimento sólido dos conceitos e soluções apresentadas pelo protocolo MIDI no armazenamento simbólico de seqüências musicais polifônicas e multi-instrumentais. Implementar novos aplicativos multimídia interativos, aplicando os conhecimentos adquiridos neste protocolo, é outro objetivo explorado neste trabalho, onde se apresenta, também, os conceitos e ferramentas multimídia utilizando uma linguagem eficiente e poderosa de última geração no paradigma funcional: a linguagem Clean. Assim, são apresentadas provas de conceitos das principais ferramentas e técnicas de programação multimídia necessárias ao desenvolvimento de tais aplicativos, permitindo-se criar interfaces visuais. O trabalho conclui por apresentar estudos de casos utilizando as bibliotecas para abertura e análise de arquivos MIDI, culminando em um aplicativo editor de partituras multitimbral e polifônico com os recursos básicos de edição utilizados em editores profissionais.

Abstract

One of the main goals of this work is to produce didactic material that allows a programmer or a musician to implement qualified multimedia applications by using a low level MIDI protocol and the SMF format 0 and 1. A chapter is dedicated to make clear all the concepts and solutions that involve MIDI in the polyphonic and multi-instrumental music. The author presents the underlying support for choosing a programming language adherent to the musical domain in the development of new applications using the foresaid technology. Sixteen factors led to the choice of the functional paradigm and of the language CLEAN to develop musical applications. The careful reader will also find proofs of concept for the foremost multimedia tools and techniques for developing graphical user interfaces. The work culminates in the presentation of case studies on how to use libraries for opening and analyzing MIDI files. Finally an editor for multi-timbral and polyphonic scores displays many features present only in professional tools.

Publicações

O autor desta dissertação é musicista, formado pela Faculdade Paulista de Artes – FPA em 2003, com especialização em Informática Educativa, pela Universidade Federal de Lavras - UFLA em 2004, durante seu trabalho de pesquisa e desenvolvimento da mesma, publicou, entre outros, os seguintes artigos e livro:

LIVROS

1. **Computação Musical – Encore 4.2.1 & Band-in-a-box 10** André Campos Machado, Luciano Vieira de Lima, Marília Mazzaro Pinto, Hércio Camargo Júnior e Júnia Helena Vieira – Editora Érica

Trabalhos completos em congressos e eventos

1. CAMARGO JÚNIOR, Hércio; LIMA, Luciano Vieira; PINHEIRO Alan Petrônio; LIMA Sandra Fernandes de Oliveira; PEREIRA, Adriano Alves. **TÉCNICAS DE CONSTRUÇÃO DE COMPILADORES APLICADAS À MANIPULAÇÃO NÃO DESTRUTIVA DO PROTOCOLO MIDI PARA GERAÇÃO DE ARQUIVOS SMF MULTISEQÜENCIAIS, COMPATÍVEIS COM OS FORMATOS GM, GS E XG, SEM A PERDA DA ESTRUTURA E EVENTOS INDIVIDUAIS ORIGINAIS - WCCSETE'2006**
2. CAMARGO JÚNIOR, Hércio; LIMA, Luciano Vieira Lima; LIMA, Sandra Fernandes de Oliveira; PEREIRA, Adriano Alves; PINHEIRO, Alan Petrônio. **A METHOD FOR PREPARING EXPERTS IN COMPUTER ENGINEERING SUBJECTS - WCCSETE'2006.**

Artigos completos publicados em revista internacional (Brasil e Portugal)

1. CAMARGO JÚNIOR, Hécio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; PEREIRA, Adriano Alves; PINHEIRO, Alan Petrônio. **BEST MERGE 2005 – COLOQUE VÁRIAS MÚSICAS EM UM SÓ ARQUIVO MIDI** .Playmusic, São Paulo, v.95, p. 7-9, ISSN/ISBN: 14151871.
2. CAMARGO JÚNIOR, Hécio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; PEREIRA, Adriano Alves; PINHEIRO, Alan Petrônio. **BEST MERGE 2005 – COLOQUE VÁRIAS MÚSICAS EM UM SÓ ARQUIVO MIDI – Parte II-** .Playmusic, São Paulo, v.96, p. 7-9, ISSN/ISBN: 14151871.
3. CAMARGO JÚNIOR, Hécio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos. **COMO CONVERTER PARA PARTITURA O QUE ESTAMOS CANTANDO OU TOCANDO AO MICROFONE DO COMPUTADOR? - AUTOSCORE**. Playmusic, São Paulo, v. 64, p. 16-18, ISSN/ISBN: 14151871.
4. CAMARGO JÚNIOR, Hécio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos. **COMO CONVERTER PARA PARTITURA O QUE ESTAMOS CANTANDO OU TOCANDO AO MICROFONE DO COMPUTADOR? - AUTOSCORE - PARTE II**. Playmusic, São Paulo, v. 65, p. 68-71, ISSN/ISBN: 14151871.
5. CAMARGO JÚNIOR, Hécio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos. **CONVERSOR POLIFONICO DE ÁUDIO PARA MIDI - AMAZINGMIDI**. Playmusic, São Paulo, v. 66, n. 1, p. 15-18, ISSN/ISBN: 14151871.
6. CAMARGO JÚNIOR, Hécio; VIEIRA, Júnia Helena; MACHADO, André Campos; LIMA, Luciano Vieira. **BAND-IN-A-BOX VERSÃO 11**. Playmusic, São Paulo, v. 52, p. 7-9, ISSN/ISBN: 14151871.
7. CAMARGO JÚNIOR, Hécio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; PEREIRA, Adriano Alves; MACHADO, André Campos. **SONAR 4 – As novidades da nova versão! (Parte II)**. Playmusic, São Paulo,

- v. 94, p. 8-10, ISSN/ISBN: 14151871.
8. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos. **BAND-IN-A-BOX 2004**. Playmusic, São Paulo, v. 85, p. 16-18, ISSN/ISBN: 14151871.
 9. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos. **Sibelius 3.1.3 – Montando exercícios de escalas e arpejos automaticamente**. Playmusic, São Paulo, v. 88, p. 16-18, ISSN/ISBN: 14151871.
 10. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos. **Determinando seu alcance vocal de conforto e adequando suas seqüências MIDI para acompanhar você e seu grupo vocal**. Playmusic, São Paulo, v. 89, p. 14-17, ISSN/ISBN: 14151871.
 11. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; PEREIRA, Adriano Alves; MACHADO, André Campos. **Best Rest 2005**. Playmusic, São Paulo, v. 92, p. 8-11, ISSN/ISBN: 14151871.
 12. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; PEREIRA, Adriano Alves; MACHADO, André Campos. **SONAR 4 – As novidades da nova versão!** Playmusic, São Paulo, v. 93, p. 8-11, ISSN/ISBN: 14151871.
 13. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; SILVA, Priscila Thays Lemos da; MACHADO, André Campos. **Band-in-a-Box 2004 – A nova versão**. Playmusic, São Paulo, v. 81, p. 10-13, ISSN/ISBN: 14151871.
 14. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos. **Analizando o DVD Architect 5.0 da Sony**. Playmusic, São Paulo, v. 82, p. 8-10, ISSN/ISBN: 14151871.
 15. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos. **Analizando o DVD Architect 5.0 da Sony – Parte II**. Playmusic, São Paulo, v. 83, p. 8-10, ISSN/ISBN:

14151871.

16. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos. **Band-in-a-Box 2004 – Criando harmonias vocais de áudio conforme tonalidade e harmonia de sua sequência.** Playmusic, São Paulo, v. 82, p. 13-15, ISSN/ISBN: 14151871.
17. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos. **Band-in-a-Box 2004 – Como afinar automaticamente sua trilha de áudio (canto e outros instrumentos acústicos), através da harmonia de seu arranjo MIDI.** Playmusic, São Paulo, v. 84, p. 14-16, ISSN/ISBN: 14151871.
18. CAMARGO JÚNIOR, Hércio; LIMA, Luciano Vieira; MACHADO, André Campos. **Band-in-a-Box Versão 10.** Playmusic, São Paulo, v. 43, p. 7-9, ISSN/ISBN: 14151871.
19. CAMARGO JÚNIOR, Hércio; LIMA, Luciano Vieira. **Convertendo estilos do Yamaha PSR-620 para o PSR-630.** Playmusic, São Paulo, v. 44, p. 10-11, ISSN/ISBN: 14151871.
20. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos; VIEIRA, Júnia Helena. **Montando seu CD com a super banda do Band-in-a-Box 10.** Playmusic, São Paulo, v. 49, p. 68-70, ISSN/ISBN: 14151871.
21. CAMARGO JÚNIOR, Hércio; LIMA, Sandra Fernandes de Oliveira; LIMA, Luciano Vieira; MACHADO, André Campos; VIEIRA, Júnia Helena. **Montando seu CD com a super banda do Band-in-a-Box 10 – Parte II.** Playmusic, São Paulo, v. 50, p. 68-70, ISSN/ISBN: 14151871.

Conteúdo

Capítulo 1 - Introdução, Objetivos, Justificativas	1
 Capítulo 2 – O padrão MIDI.....	6
2.1 Conector e cabo MIDI.....	8
2.2 Tipos de conexões MIDI entre teclados.....	9
2.3 Conexão entre teclado e computadores.....	11
2.4 O Protocolo MIDI.	14
2.5 Execução em Tempo Real.....	16
2.6 Mensagens MIDI.....	16
2.6.1 Formato do comando de ativar notas.	18
2.6.2 Mensagens MIDI e número de bytes por mensagens.....	21
2.6.3 Ativando e desativando uma nota musical.	22
2.6.4 Byte de dados com 8 bits: uma exceção à regra geral.....	25
2.6.5 O Contador MIDI de 1 a 4 Bytes.	26
2.6.5.1 Regra para se determinar o número de bytes de um contador MIDI.	29
2.6.5.2 Fórmulas para se converter uma contagem de 1 a 4 Bytes em um inteiro equivalente.	30
2.7 Classificação das mensagens quanto ao número de bytes.	30
2.8 Resumindo o envio de mensagens em tempo real.	31
2.9 Arquivos MIDI padrão: Os Standard MIDI Files (SMF).	33
2.9.1 A máquina MIDI.	33
2.9.2 Como a máquina MIDI grava e lê mensagens em SMF.	34
2.9.2.1 Gravando mensagens em Standard MIDI Files.....	34
2.9.2.2 Ativando e desativando notas em um SMF.....	38
2.9.2.3 Delta Time.	39
2.9.2.4 Contagem MIDI x tempo cronológico.	39

2.9.2.5	Tempo de uma nota musical em uma partitura.	39
2.9.2.6	Tempo de um evento em um arquivo SMF.	41
2.9.2.7	Ppq.	44
2.9.2.8	O cálculo do tempo de um evento em segundos.	45
2.9.2.9	Meta Evento Set Tempo.	46
2.9.2.10	Lendo uma mensagem com a máquina MIDI.	47
2.9.2.11	O Contador regressivo.	48
2.10	Diferença básica entre SMF formato 0 e formato 1.	49
2.10.1	Formato 0.	50
2.10.2	Formato 1.	51
2.11	O Arquivo MIDI SMF.	53
2.11.1	Cabeçalho Principal (Header Chunk).	54
2.11.2	Trilhas de mensagens (Track Chunk).	56
2.11.3	Estrutura da trilha de mensagem do Formato 0.	56
2.11.4	Estrutura das trilhas de mensagens do Formato 1.	57
2.12	Exemplo de uma seqüência musical a ser registrada em arquivo MIDI SMF	
	Formato 0 e Formato 1.	58
2.12.1	Arquivo MIDI Formato 0 da figura 2.57, sem <i>running status</i>	59
2.12.2	Arquivo MIDI Formato 0 da figura 2.58, com <i>running status</i>	66
2.12.3	Arquivo MIDI Formato 1 da mesma figura 2.58.	70
2.13	Conclusão.	81
 Capítulo 3 - Linguagem Funcional Clean e Implementações Básicas		85
3.1	A Escolha pela Linguagem Clean.	85
3.2	Aderência e Notação Zermelo-Fraenkel: Notas da escala de dó maior.	88
3.2.1	Implementando uma função em Clean que testa se o último caractere de uma palavra, de uma <i>string</i> , é ou não um acidente musical.	90
3.3	Aderência e Notação Zermelo-Fraenkel: Tríades musicais.	92
3.3.1	Tríades Maiores.	93
3.3.2	Tríades menores.	95
3.4	Implementações básicas.	95
3.4.1	Exemplo da função <i>StringTokens</i> aplicada ao domínio musical : Transpondo uma música em formato texto.	96

3.4.2 Transformando uma lista de <i>strings</i> em uma <i>string</i> com um separador entre os elementos da lista pré-especificado.	104
3.5 Conclusão.....	108
<u>Capítulo 4 - Bibliotecas para manipulação de arquivos MIDI</u>	109
4.1 Biblioteca <i>separaEventosF0</i>	109
4.1.1 Analisando se o evento é um evento de ativar nota com delta time de um Byte.	114
4.1.2 Analisando se o evento é um Meta Evento qualquer com delta time de um byte.	115
4.1.3 Outras funções da biblioteca.	116
4.2 Lendo eventos de arquivos MIDI SMF formato 1.	118
4.3 Gera MIDI formato 0.	119
4.4 Gera MIDI formato 1.	125
4.5 Conclusão.....	128
<u>Capítulo 5 - Editor Didático de Partituras</u>	129
5.1 Controle interativo variável de popup através de ícones da barra de ferramentas..	130
5.1.1 Status nota/acorde.	130
5.1.2 Escolha de instrumento.	131
5.1.3 Volume.	132
5.1.4 Figuras musicais.....	133
5.2 Atualização da janela (refresh).....	134
5.3 <i>Plotar</i> nota musical.	135
5.4 Apagar nota musical	136
5.5 Parâmetros iniciais.	139
5.6 Conclusão.....	140
<u>Capítulo 6 - Validação e Estudos de Casos</u>	141
6.1 Aplicando a biblioteca <i>separaEventosF0</i> na identificação e ações em arquivos MIDI formato 0.	142
6.1.1 – Consultas das notas musicais pertencentes a um SMF formato 0.....	143

6.1.2 - Devolvendo nomes das notas musicais ao invés dos códigos MIDI.	146
6.1.3 Determinando o range de um canal MIDI especificado.....	148
6.1.4 Consulta sobre as mensagens de meta eventos.	150
6.1.5 Transformação – Transpondo uma música de um SMF.	151
6.1.6 - Implementando o aplicativo que faz a transposição de uma música em semitons, conforme explicado anteriormente:	154
6.2 Aplicando as bibliotecas geraMidiF0 para validação do exemplo dado no item 2.12 do Capítulo 2.....	158
6.2.1 - Arquivo MIDI Obtido Teoricamente, Conforme Capítulo 2, Item 2.12.	158
6.2.2 Arquivo MIDI obtido pela função geraMidiF0 implementada nesta dissertação.	159
6.3 Aplicando as bibliotecas geraMidiF1 para validação do exemplo dado no item 2.12 do Capítulo 2.....	162
6.3.1 Arquivo MIDI Obtido Teoricamente, Conforme Capítulo 2, Item 2.12..	162
6.3.2 Arquivo MIDI Obtido pela Função geraMidiF1 Implementada nesta Dissertação.	162
6.4 Implementação de um Leitor MIDI formato 0 ou 1 e Conversor Didático de arquivos MIDI escritos em Hexadecimal e Decimal para SMF formato 0 ou 1.....	168
6.4.1 Potencialidades das ferramentas da interface.....	169
6.4.2 PopUp 1 (Texto): Ações.....	169
6.4.3 PopUp 2 (MIDI) -Ações:.....	170
6.4.4 PopUp 3 (PLAYER): Ações.	171
6.4.5 PopUp 4 (TOOLS): Ações	171
6.4.6 Calculadora: Delta Time <-> PPQ.	173
6.4.6.1- Convertendo ppq para DeltaTime.	173
6.4.6.2 - Convertendo Delta Time para ppq (pulsos).	173
6.5 Implementação de um Editor de Partituras Multi-Instrumental e Multi-Canal. Com salvamento em formato proprietário e em SMF formato 1.....	175
6.5.1 Descrição dos componentes deste editor:.....	177
6.6 Conclusão.....	191

<u>Capítulo 7 - Conclusão e trabalhos futuros</u>	193
7.1 Conclusão.....	193
7.2 Trabalhos futuros.	195
<u>Referências</u>	197
<u>Anexo I - Monotimbralidade</u>	200
<u>Anexo II – Especificação MIDI 1.0</u>	201
<u>Anexo III - Tonalidades</u>	219
<u>Anexo IV – Tipos de Mensagens MIDI</u>	220
<u>Anexo V - Valores Máximo e Mínimo de cada contagem com 1, 2, 3 e 4 Bytes</u>	222
<u>Anexo VI – Figuras Musicais</u>	224
<u>Anexo VII - Polimorfismo, Instâncias e Sobrecarregamento</u>	226
<u>Anexo VIII - Benchmark</u>	240
<u>Anexo IX – Utilizando Clean pela primeira vez</u>	242
Programando com a técnica de Tipos Únicos	247
Abrindo, manipulando e salvando Arquivos.....	250
Leitura de Arquivo de Dados	251
Escrita em Arquivo de dados	253
Criando uma biblioteca	254
Separando <i>Tokens</i> de uma <i>String</i> ou de um arquivo texto	257
<u>Anexo X - Interfaces Visuais e Multimídia</u>	260
Interface NDI	261
Interface SDI	265

Interface SDI com Menus	268
Menu Opções	269
Menu Help.....	272
Interface SDI com Botões, Campo de Texto, Texto, Radio Button, Check Button e Pop-Up.....	273
Atributos da janela	275
Componentes da Interface.....	277
Botão	277
Campo de Edição de Texto	280
Botão tipo <i>Radio</i>	281
Botão tipo Check.....	281
Menu Popup	282
Texto (campo de texto)	282
Escrevendo em um campo de texto (editável ou não).....	283
Lendo um campo de texto	284
Registro Local em interface SDI.....	284
Exemplo de ToolBar	291
Exemplo de Plotar Bitmap	294
Interface para leitura e reconhecimento de Eventos de Mouse.....	299
Teclado Virtual.....	305
Criar Pentagrama.....	310

Anexo XI - Como Inserir ícone na barra de título de um programa executável

compilado em Clean.....	316
Compilar a dll a cada mudança de ícone:.....	316
Alterar o programa .icl – fonte em Clean.....	318
Criando um instalador com o Create Install Free.....	319
Interface principal do programa.	320
Procedimentos	320

<u>Índice Remissivo</u>	322
--------------------------------------	-----

Lista de Acrônimos

MIDI	- Musical Instruments Digital Interface
NAMM	- National Association of Music Merchants
USI	- Universal Synthesizer Interface
MPU-401	- MIDI Processing Unit
USB	- Universal Serial Bus
MSB	- Most Significant bit
LSB	- Least Significant bit
SMF	- Standard MIDI Files
CPN	- Common Practice Notation – uma partitura tradicional
Ppq	- Pulses per quarter note
SMPTE	- Society of Motion Picture and Television Engineers – Padrão de contagem de tempo utilizada para sincronismo de som com imagem
ASCII	- American Standard Code for Information Interchange

Capítulo 1

Introdução, Objetivos, Justificativas.

Atualmente se tem um número expressivo de aplicativos multimídia destinados à resolução de problemas no domínio musical, tanto em plataformas proprietárias quanto em arquitetura e código aberto. Uma das tecnologias agraciadas com os recursos eletrônicos e computacionais mais modernos é a tecnologia MIDI¹, tanto em equipamentos voltados a uma completa automação de shows, envolvendo iluminação, equalização, mixagem, mesas de som inteligentes, hardware para afinação vocal e *swap* de *play-back*, quanto em *software* também para correção de afinação e voz em estúdio, seqüenciamento musical áudio e MIDI, edição de partituras tradicionais convencionais e modernas, arranjo automático, digitalização de partituras, síntese virtual, assistência à composição e outras potencialidades mais.

Ao se ingressar neste universo de opções e recursos tecnológicos um mundo novo de possibilidades se abre ao profissional e *hobista* em música, em engenharia eletrônica, engenharia de som e ao profissional em computação. Surge daí uma nova área do conhecimento: a Computação Musical ou Sônica (no Brasil), interagindo, também, com outras áreas do conhecimento, tais como a filosofia, lingüística, biomédica e a psicologia, permitindo ao homem ultrapassar suas limitações ligando a inteligência artificial e as técnicas de construção de sistemas especialistas para tentar dotar as máquinas, o computador, de capacidades e habilidades onde apenas a criatividade e os dons humanos ousavam adentrar.

Apaixonado por este complexo domínio multidisciplinar, e tendo cursado disciplinas em curso de especialização e computação musical na UFU, nasceu no autor desta dissertação o desejo de se especializar não apenas na utilização dos recursos

¹ MIDI – Musical Instruments Digital Interface

tecnológicos existentes até então, mas participar do processo de geração de novos produtos e novos conhecimentos.

Apesar de, a princípio, parecer já existir todos os possíveis e imagináveis aplicativos computacionais e equipamentos utilizando a tecnologia MIDI nos processos musicais, sempre surge uma nova idéia de um novo produto ou de uma solução mais simples e aderente de se resolver um problema.

Um exemplo típico de um aplicativo que motivou o início dos estudos deste trabalho foi o desejo e a necessidade de se colocar várias seqüências musicais em um único arquivo MIDI para play-back. Para tanto, existe o formato de arquivo MIDI SMF (Standard MIDI Files) formato 2, o qual seria destinado a esta função. Infelizmente o mesmo não foi adotado como padrão pelos equipamentos e programas existentes da atualidade. Desta forma, esta ação é uma tarefa que atualmente tem sido feita manualmente utilizando programas de seqüenciamento, como, por exemplo, o Sonar ou o Pro Tools. O que ocorre é que este é um processo complexo e demorado, demandando do usuário, do músico, um conhecimento profundo do protocolo MIDI (ver capítulo 2) e de manipulação dos softwares de edição e seqüenciamento musical. Outros aplicativos, como, por exemplo, eliminar pausas no início e fim de arquivo, composição automática extraindo conhecimentos direto de arquivos MIDI, editores de partitura com dinâmica humanizada a cada nota, são projetos que também incrementaram o autor a se especializar neste protocolo e no estudo de técnicas computacionais multimídia e em interfaces visuais para poder implementar tais projetos e outros que os irão suceder.

Objetivo.

Produzir um material didático que permita ao leitor trabalhar com segurança no protocolo MIDI, em baixo nível, bem como implementar bibliotecas computacionais que permitam ler, editar e modificar arquivos MIDI SMF. Para tanto, deve-se definir um paradigma e uma linguagem de programação que seja aderente ao domínio musical e que permita implementar aplicativos multimídia interativos utilizando interfaces visuais legíveis. Objetiva-se, assim, implementar, criar bibliotecas e aplicativos que exemplifiquem e concretizem os conceitos dos principais recursos multimídia, tais como:

1. Construção de bibliotecas de funções para se ler, editar e salvar arquivos MIDI SMF formato 0 e formato 1;

2. Implementar uma biblioteca para se poder reproduzir arquivos MIDI;
3. Produzir um material didático para o ensino do protocolo MIDI e dos SMF formato 0 e formato 1;
4. Definir e justificar a escolha de um paradigma e de uma linguagem de programação para implementação de sistemas multimídia interativos com interfaces visuais aderentes ao domínio musical com baixa relação custo/aprendizado;
5. Apresentar os conceitos e implementar exemplo aplicativo de construção de interfaces NDI e SDI²;
6. Apresentar os conceitos e implementar interfaces SDI com botões, campos de textos, *pop-ups*, *radio buttons*, *check buttons*;
7. Apresentar os conceitos e implementar interfaces SDI com menus e barra de ferramentas;
8. Apresentar os conceitos e implementar um teclado virtual: *plotagem* de *bitmaps* em janelas e leitura de posicionamento e ações de mouse;
9. Apresentar os conceitos e implementar editores visuais de música multicanal e multitimbral;
10. Apresentar os conceitos e implementar um kit didático para o ensino e construção de arquivos MIDI em baixo nível (em hexadecimal e decimal);
11. Apresentar os conceitos e implementar soluções para inclusão de ícones nos arcabouços dos aplicativos gerados;
12. Mostrar como criar um projeto de instalação automática dos aplicativos desenvolvidos;
13. Promover estudos de casos e validações dos conceitos, bibliotecas e aplicativos gerados neste trabalho.

Efetivados tais objetivos, o leitor deverá estar apto a desenvolver seus próprios aplicativos multimídia interativos utilizando o protocolo MIDI e SMF.

No capítulo 1 é abordada uma visão geral da abrangência e relevância da utilização do protocolo MIDI e dos arquivos SMF (Standard MIDI Files). Também é apresentada a motivação que levou à efetivação deste trabalho. Assim, conclui-se pela importância de

² NDI – No Document Interface, SDI – Single Document Interface

se conhecer e dominar os conceitos e técnicas que levem ao entendimento e manipulação de tal protocolo e seus arquivos. Desta forma, podem-se desenvolver aplicativos MIDI que apresentem novas soluções para diversos problemas ainda em aberto na automação de processos no domínio musical.

O capítulo 2 aborda o protocolo dos arquivos MIDI (SMF) formato 0, o qual é aceito universalmente por todos os seqüenciadores de hardware e software. Em seqüência são apresentados algoritmos esquemáticos de como manipular tais arquivos para extração dos conhecimentos (eventos) existentes nos mesmos.

O capítulo 3 apresenta a linguagem funcional CLEAN, justificando o porquê utilizar a mesma para desenvolver aplicativos MIDI. Neste capítulo algumas funções dedicadas à manipulação de conteúdo multimídia são desenvolvidas para que se possa utilizar CLEAN no desenvolvimento de aplicativos MIDI a serem utilizadas no capítulo 4.

O capítulo 4 é responsável por apresentar técnicas e soluções para o desenvolvimento de aplicativos MIDI em formato 0 utilizando a linguagem funcional CLEAN. Partindo das funções primitivas e implementadas no capítulo 3, aborda-se com detalhes a implementação de diversas funções e bibliotecas que permitem a manipulação e edição de eventos MIDI, tais como:

- Abrir o arquivo MIDI e converter para um lista de inteiros;
- Analisar e separar cada evento;
- Efetuar transformações em eventos;
- Salvar o arquivo MIDI;
- Gerar arquivos MIDI formato 0 e 1, a partir de listas pré-formatadas;
- Tocar os arquivos MIDI, etc.

O capítulo 5 apresenta a implementação de vários aplicativos MIDI, com interfaces visuais NDI e SDI, de tal forma a permitir que se criem aplicativos multimídia interativos nesta linguagem a partir dos conceitos básicos e fundamentais, abrangendo desde a manipulação básica dos eventos até a colocação de ícones no arcabouço, e criação de instalador automático. O aplicativo final é um editor didático de partituras que abrange os conceitos e implementações relatados no capítulo 5.

O capítulo 6 apresenta a validação e estudos de casos, empregando as bibliotecas criadas nos capítulos anteriores, bem como os conceitos de interface, gerando novos aplicativos multimídia aplicados ao domínio musical.

O Capítulo 7 apresenta a conclusão final avaliando a concretização dos objetivos propostos e apresentando sugestões para trabalhos futuros.

Capítulo 2

O padrão MIDI.

Na feira do **NAMM** (National Association of Music Merchants) de 1981, local onde se encontram profissionais das mais expressivas empresas produtora de tecnologia e equipamentos de áudio e musicais, três dos maiores fabricantes de equipamentos musicais, principalmente teclados, a Roland Corporation, a Sequential Circuits e a Oberheim Eletronics, se reuniram para buscas uma solução a um problema emergente que começava a causar transtornos técnicos aos músicos de bandas que estavam introduzindo teclados eletrônicos em suas apresentações. Este problema era o de como interligar vários teclados de mesma marca ou de marcas diferentes para que um único músico pudesse controlá-los ao mesmo tempo, a partir de um único teclado. As soluções proprietárias demandavam um número excessivo de cabos para interconexão entre os equipamentos, gerando mal contato nos cabos e conectores, o que exigia ter um bom conhecimento técnico em eletrônica pelos músicos e pelos seus assistentes. O problema se agravava quando a interconexão se dava entre teclados de fabricantes diferentes, exigindo dos técnicos a necessidade de se conhecer também a fundo os circuitos eletrônicos de cada equipamento, de cada fabricante.



Figura 2.1 – Ilustração da interconexão de teclados

A necessidade de se agrupar vários teclados se dava principalmente devido à maioria dos existentes na época, serem monofônicos³ e monotimbrais⁴ [1], [2]. Outro motivo era, e ainda é até hoje, devido ao fato de que cada fabricante, cada linha de teclados, produzir, sintetizar melhor alguns timbres do que outros. Como exemplo, pode-se citar a Korg nos timbres de cordas e a Roland nos timbres de piano. Deste grupo, surgiu a idéia de se criar um protocolo⁵ padrão de comunicação entre teclados musicais eletrônicos, permitindo que se pudesse conectá-los sem a necessidade de um conhecimento dos circuitos eletrônicos de cada fabricante e das reais potencialidades de cada teclado. Uma vez concretizado o projeto e implementação deste padrão de comunicação, cada fabricante se encarregaria de inserir em seus produtos, uma interface que realizasse o processo de comunicação entre seus teclados e outros que também possuísem a interface padrão.

Desta idéia e objetivo, surgiu a proposta do protocolo, denominado pelos mesmos de Protocolo **MIDI** (**M**usical **I**nstruments **D**igital **I**nterface) e a interface, também padrão, a **USI** (**U**niversal **S**ynthesizer **I**nterface).

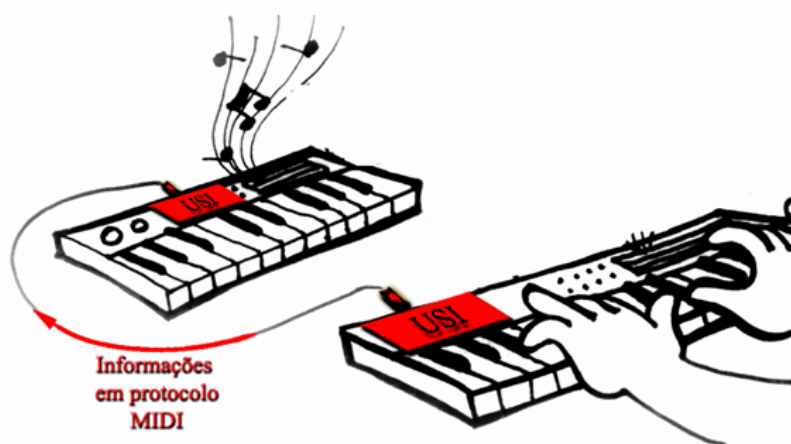


Figura 2.2 – Ilustração de uma conexão para o envio de informações em protocolo MIDI

³ Monofônico – a limitação de um equipamento apenas conseguir gerar uma única nota musical de cada vez.

⁴ Monotimbralidade – a limitação de um equipamento de produzir (sintetizar) apenas um timbre musical de cada vez, onde timbre, é a característica de um som que nos faz reconhecê-lo e diferenciá-lo dos demais sons (ver Anexo I).

⁵ Protocolo – um conjunto de procedimentos que devem ser seguidos para que a informação possa ser entendida e executada corretamente.

Assim, depois de dois anos de reuniões e testes, no **NAMM** de janeiro de 1983, estas empresas apresentaram ao público a **MIDI Specification 1.0** (ver Anexo II), a qual continha todos os detalhes do protocolo MIDI e respectiva interface USI de comunicação. Essa especificação foi tão bem elaborada que até nos dias de hoje ela se mantém íntegra, válida, apesar dos avanços sensíveis da tecnologia, sofrendo apenas alguns acréscimos já previstos no projeto inicial.

2.1 Conector e cabo MIDI.

Para que um controlador (*master*) pudesse enviar suas mensagens para outro teclado (*slave*), utilizando uma interface USI, foram idealizados conectores e cabos que fossem fáceis de serem encontrados ou de fácil manutenção em qualquer lugar onde uma banda fosse tocar. Desta forma, adotou-se no projeto o conector DIM de 5 pinos, por ser o mesmo bastante comum e fácil de ser encontrado em qualquer empresa de manutenção de equipamentos de áudio. Apesar deste conector possuir 5 pinos, apenas 3 dos mesmos foram utilizados (pinos 5, 2 e 4), ficando os outros dois (1 e 3) como reserva ou para utilização na comunicação extra entre teclados de mesmo fabricante.

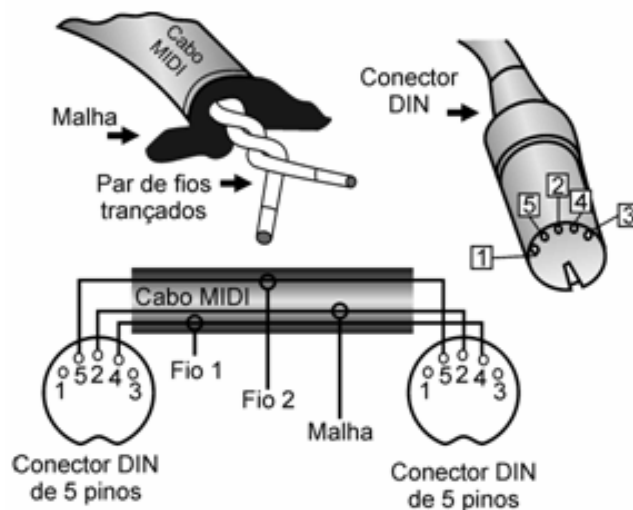


Figura 2.3 – Conector e cabo MIDI

2.2 Tipos de conexões MIDI entre teclados.

O teclado controlador (*master*) possui, então, uma conexão denominada MIDI OUT para enviar as mensagens para outro teclado controlado (*slave*), onde existe uma conexão denominada MIDI IN para receber as mensagens e comandos.

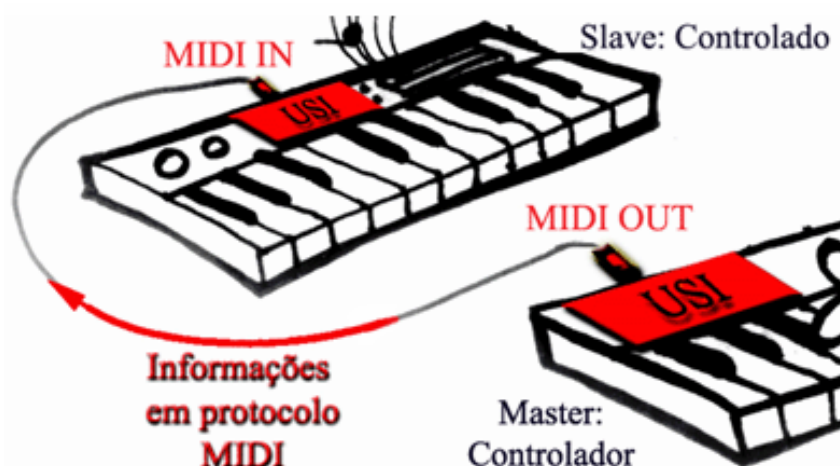


Figura 2.4 – Ilustração de uma conexão **MIDI IN** e **MIDI OUT**

Como não fazia e nem faz sentido se ter uma interface apenas para enviar mensagens e outra apenas para receber mensagens, a interface padrão deveria possuir pelo menos as duas opções de conexão: MIDI IN e MIDI OUT. Estas duas conexões permitiriam que qualquer teclado viesse a ser o controlador.

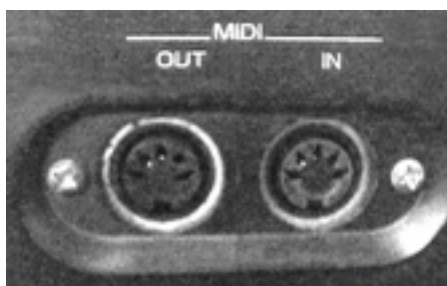


Figura 2.5 – Conexão **MIDI IN** e **MIDI OUT**

Como muitos dos teclados existentes eram monotimbrais, existia a necessidade de se ter mais de um teclado, cada um reproduzindo um determinado instrumento musical, ou seja, um músico poderia desejar ao tocar em seu teclado, que o som produzido fosse de piano com *strings* (violino, violoncelo, etc.), em alguns casos, até mais de dois timbres ao mesmo tempo. Como a interface possui geralmente⁶ apenas uma conexão MIDI

⁶ Atualmente existem placas e interfaces com mais de uma conexão MIDI OUT E MIDI IN.

OUT, idealizou-se mais um tipo de conexão na mesma: a conexão MIDI THRU. A figura 2.6 mostra esta conexão junto com as duas precedentes.

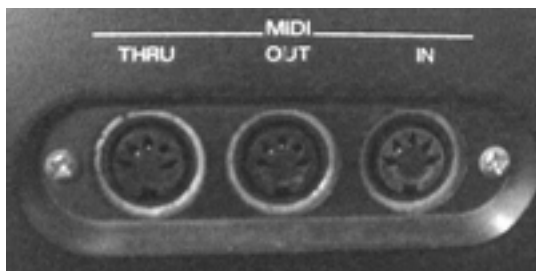


Figura 2.6 – Conector MIDI THRU MIDI OUT E MIDI IN

Esta conexão MIDI THRU, permite a um teclado controlado (*slave*) enviar uma cópia da mensagem recebida do teclado controlador para um terceiro teclado controlado (*slave*). Desta forma, o teclado controlador passa a poder controlar mais de um teclado escravo.



Figura 2.7 – Conectores e conexões MIDI IN, MIDI OUT E MIDI THRU

A figura 2.8 ilustra um teclado controlador (*master*) controlando dois teclados escravos (*slaves*) ao mesmo tempo.

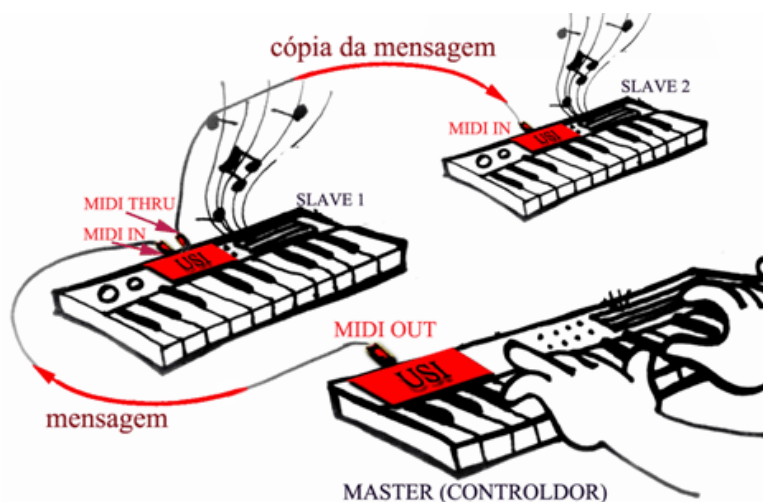


Figura 2.8 – Ilustração das ligações MIDI OUT, MIDI IN E MIDI THRU

Mais detalhes de vários tipos de interligação entre teclados *master* e *slaves* podem ser encontradas na dissertação de MACHADO [1] e no capítulo 17 do livro Cakewalk Sonar 2.0 [8].

2.3 Conexão entre teclado e computadores.

Vislumbrando que os computadores logo adotariam o padrão MIDI para controlar processos musicais e para reproduzir e enviar músicas a outros equipamentos MIDI, em 1984 a *Roland Corporation* lançou no mercado uma placa de interface MIDI denominada MPU401 [7]. A mesma foi logo utilizada pelos computadores Amiga e Macintosh, ampliando os horizontes dos músicos, não apenas na automação de processos musicais de execução ao vivo (shows), como nos processos de assistência à composição musical, editoração de partituras, arranjos e outros mais. Apenas em 1987 os computadores da linha PCs da IBM e compatíveis começaram timidamente a utilizar este padrão. Hoje em dia todo sistema de música profissional (sintetizadores, *samplers*, mesas de som, mesa de luz, outros), bem como os computadores pessoais e internet utilizam o padrão MIDI (interface e protocolo). Os que não utilizam a interface física emulam as mesmas por software (criam uma máquina MPU401 virtual).

Para utilizar MIDI em um computador, necessita-se, portanto, de uma placa MPU401 ou de uma placa de som⁷ que possua esta interface incorporada (seja por hardware ou software). Além desta placa, é necessário um cabo MIDI especial⁸ com um circuito eletrônico interno com acopladores óticos⁹, para realizar a conexão da mesma com outros equipamentos MIDI. A figura 2.9 mostra o esquema eletrônico de como implementar um cabo MIDI para computador, padrão MPU401.

⁷ Uma das placas mais populares de som para computador com interface MIDI é a *Sound Blaster* da *Creative Labs* (<http://www.creative.com/>). A mesma é popular por possuir um excelente custo/benefício.

⁸ Estes cabos não apenas possuem conectores e fios, os mesmos possuem um circuito interno com acopladores óticos para isolar eletricamente os equipamentos interconectados por eles.

⁹ Com este tipo de acopladores, evita-se a queima de um dos equipamentos interligados, caso um deles, ou algum dos fios, entrem em curto circuito.

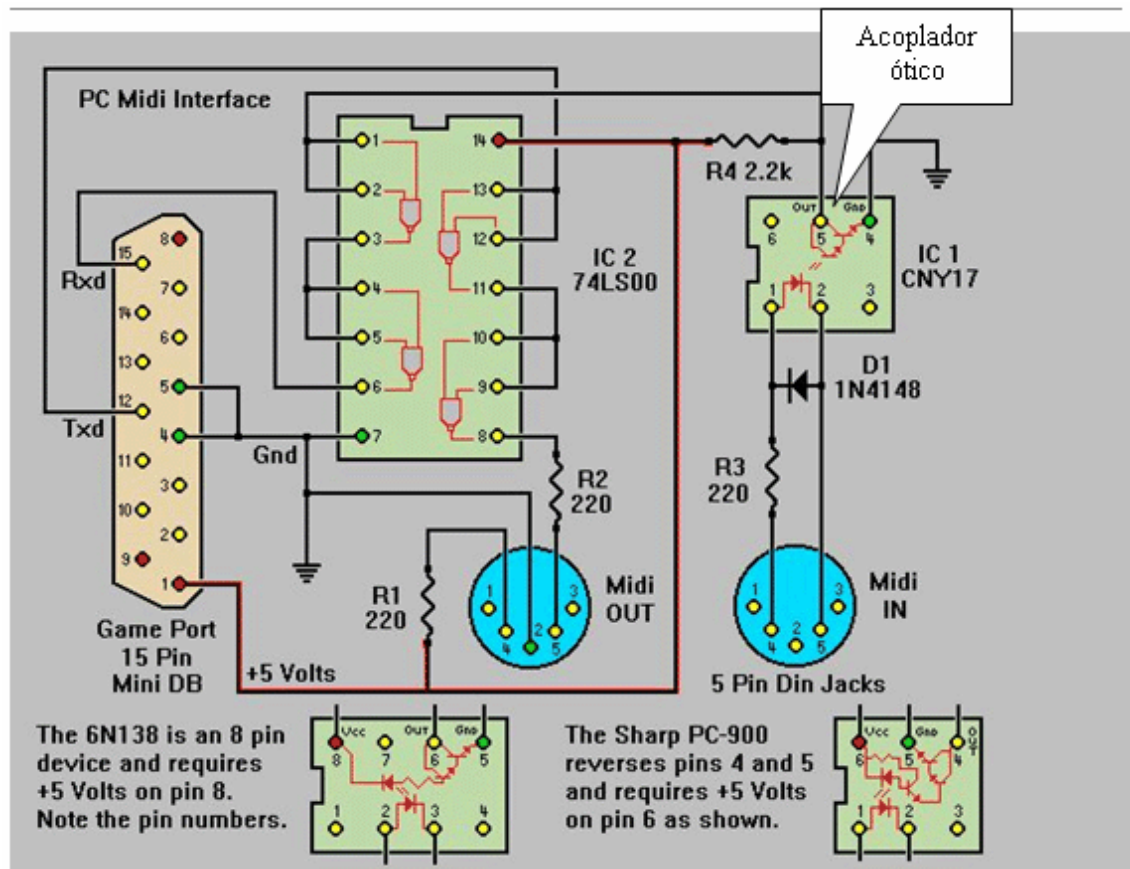


Figura 2.9 – Esquema eletrônico de um cabo MIDI padrão MPU401

As placas de som para computador, até o final do século XX, utilizavam um conector do tipo DB15 (um conector de 15 pinos tipo DB), o qual servia tanto para conexão com equipamentos MIDI quanto para conexão de um *joystick* utilizado em jogos por computador.

A figura 2.10 mostra uma placa comercial da *Creative Labs*, uma *Sound Blaster Live*, com respectivo conector.

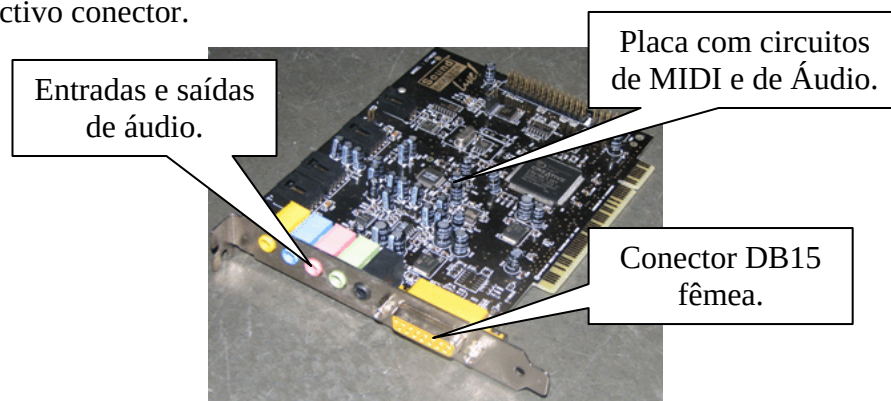


Figura 2.10 – Placa de som com conector MIDI DB15

A **figura 2.11** mostra um cabo MIDI padrão MPU401 fabricado no Brasil.



Figura 2.11 – Cabo MIDI padrão MPU401 com conexão DB15

A figura 2.12 mostra este cabo conectado na placa de som.

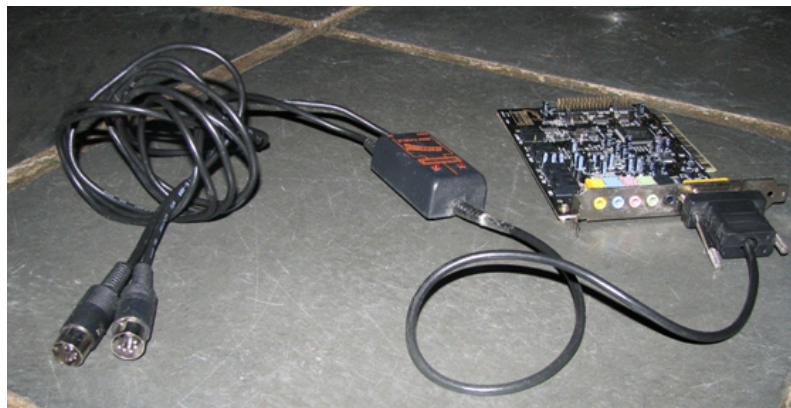


Figura 2.12 – Cabo MIDI padrão MPU401 conectado na placa de som,

Atualmente com a tecnologia USB¹⁰ para transferência de dados em alta velocidade, os *joysticks* e a maioria das placas de som MIDI são fabricadas sem os conectores DB15. As placas de som agora, possuem geralmente apenas conexões para entradas e saídas de áudio, deixando a cargo dos novos cabos MIDI USB fazerem as conexões e comunicações necessárias entre os equipamentos MIDI. As figuras 2.13 e 2.14 mostram uma placa MIDI do tipo *Sound Blaster* sem conector DB15 e um cabo MIDI USB utilizados atualmente.

¹⁰ USB – Universal Serial Bus – é um tipo de conexão *Plug and Play* que permite a conexão de periféricos sem a necessidade de desligar o computador.

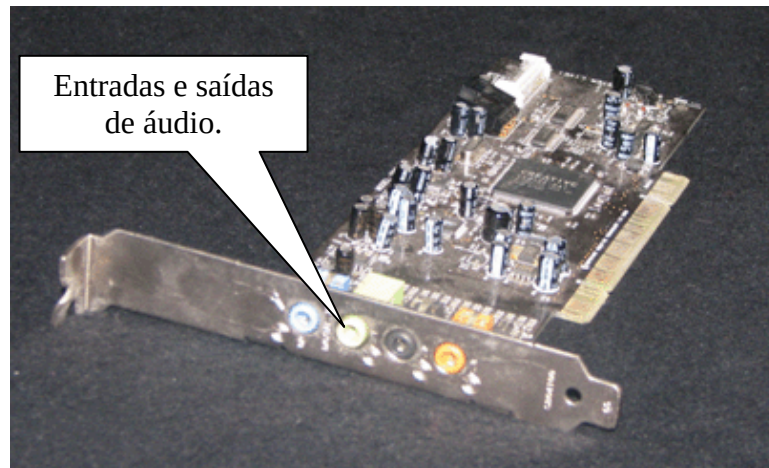


Figura 2.13 – Placa de som MIDI sem conector DB15



Figura 2.14 – Cabo MIDI USB

2.4 O Protocolo MIDI.

O protocolo MIDI inicialmente tinha como finalidade principal auxiliar o músico em apresentações ao vivo. Neste caso, o sistema deveria apenas enviar comandos de um teclado para outro em tempo real (em tempo de execução) para que o teclado controlado pudesse executar a ação solicitada. Entre estes comandos, os mais simples seriam de ativar uma nota musical quando uma tecla do controlador fosse pressionada e desativar esta mesma nota quando a mesma fosse liberada.

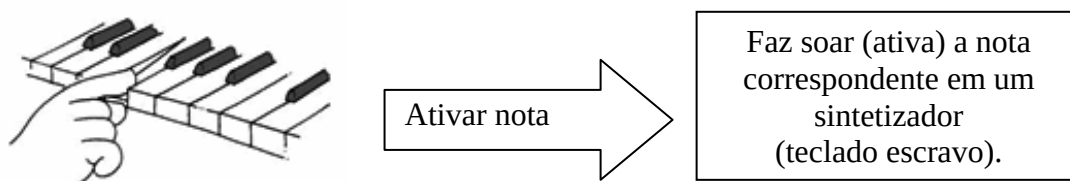


Figura 2.15 – Nota Ativada

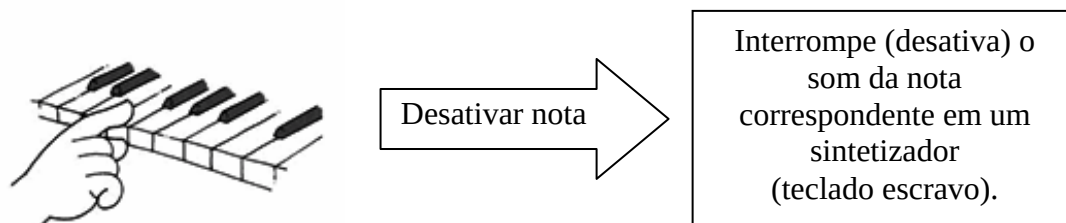


Figura 2.16 – Nota Desativada

Observe que MIDI não envia sinais sonoros, apenas comandos de um dispositivo para o outro, informando a ação que se deseja ver realizada no equipamento controlado. Dentre as várias ações que se pode enviar de um controlador para seus escravos, estão os comandos de: ativar e desativar notas, alteração de volume, *pitch bend*¹¹, mudança de instrumento e outros mais. Assim, quem é responsável pela produção final do som e respectiva dinâmica será o equipamento controlado (teclado com timbres, sintetizadores, *samplers* ou módulos de som). Desta forma, a cada ação executada no controlador, a mesma é reproduzida em tempo real nos escravos a ele associado.

O protocolo MIDI se encarrega de organizar como estes dados serão transmitidos do controlador aos escravos. As mensagens são enviadas serialmente, ou seja, bit a bit e montadas no escravo para sua execução. Desta forma, o escravo deve saber quando uma mensagem é iniciada e quando ela termina. Daí a necessidade de um protocolo, ou seja, da receita de como enviar e receber corretamente as mensagens contendo os eventos, comandos, que se quer propagar do controlador para os demais equipamentos MIDI a ele associado.

¹¹ *Pitch bend* – controle de variação de frequência da nota musical. Geralmente acionado por uma alavanca que indica o quanto se deseja alterar a frequência da nota, simulando, por exemplo, a alavanca de uma guitarra que ao esticar ou afrouxar uma corda da mesma, faz com que o som fique variando em torno da frequência real da nota tocada. Se aplicado ao som de um piano, o som produzido seria um efeito do tipo glissando, ou seja, mudando não a frequência, mas notas em torno da que foi acionada.

2.5 Execução em Tempo Real.

Conforme dito anteriormente, o padrão MIDI trabalha com transmissão serial¹² de dados, onde cada informação é composta de **no mínimo 8 bits de dados (um Byte)** ou múltiplos deste número. O Byte era a unidade computacional padrão utilizada pelos processadores dos microprocessadores¹³ da época da criação do padrão MIDI.

Para enviar estes Bytes de comandos, era necessário informar ao escravo quando a informação estava iniciando e quando a mesma estava terminando. Para tanto, adotou-se incluir no envio dos Bytes, mais dois bits: um para informar o início da mensagem (**start bit = 0**), e um para informar o fim da mensagem (**stop bit = 1**).

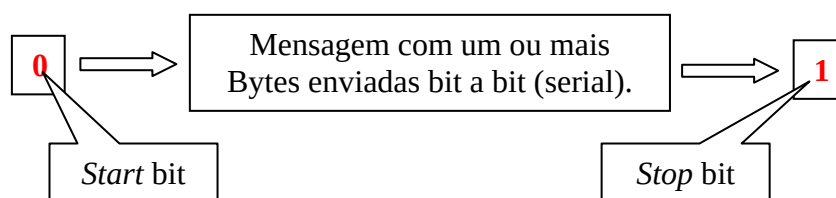


Figura 2.17 – Ilustração do “Start bit” e “Stop bit”

O que garante que o receptor, o escravo, não vai perder bits de informação, é a taxa de transmissão de dados adotada pelo padrão MIDI, devendo assim, estarem o controlador e seus escravos, sincronizados. Esta taxa de transmissão é de 31.250 bits por segundo (bit/seg = baud).

2.6 Mensagens MIDI.

Como foi dito, MIDI trabalha com a unidade Byte (8 bits) em suas mensagens. Com oito bits, pode-se discriminar 256 valores diferentes. A cada valor pode-se atribuir um tipo de ação específica.

Imagine a ação de se ativar uma nota. Fica assim a seguinte questão:

- Como o protocolo MIDI representa esta ação?

O ato de ativar uma nota demanda os seguintes conhecimentos:

¹² Serial: Envio de informação bit a bit, ou seja, dado um Byte com 8 bits, não se transmitem os mesmos simultaneamente, o que demandaria 8 fios para transmiti-los ao mesmo tempo, assim, economiza-se em fios e circuitos enviando o Byte bit a bit (serializado).

¹³ Microprocessador: um circuito eletrônico que possui uma unidade aritmética, uma unidade lógica e portas de entrada e saída de dados.

1. Qual nota deve ser ativada;
2. Qual o volume da mesma;
3. Quanto tempo a mesma deverá soar.

Na execução em tempo real, o terceiro quesito, tempo em que a nota deverá soar, é desnecessário. Diz-se isto devido a ação de interromper o som da nota, estar vinculada ao ato de liberar a tecla pressionada. Assim, basta apenas ativar a nota quando uma tecla for pressionada e desativar a nota quando a mesma for liberada. O tempo de duração da mesma é o tempo decorrido entre duas ações (ativar e desativar).

Para responder como o protocolo MIDI representa a ação de ativar uma nota, deve-se conhecer quantos Bytes são necessários para enviar uma informação de ativar uma nota. Para tanto, deve-se ter em mente que o ser humano escuta sons na faixa de frequência entre 19 a 22.000 Hz, o que, em notas musicais, dá algo em torno de 122 notas musicais. Para representar estas notas digitalmente (no sistema binário), demandaria no mínimo 7 bits ($2^7 = 128$ combinações). Assim, para representar uma nota utiliza-se um Byte de informação, já que, conforme já especificado, cada informação é representada no protocolo MIDI com no mínimo 8 bits. Para representar o volume com que esta nota será ativada, adotou-se também, utilizar um Byte. O valor do volume da nota é associado à velocidade com que a tecla foi pressionada¹⁴. Os teclados atuais normalmente são sensíveis¹⁵. Nos teclados que não são sensíveis, o volume é definido previamente em um controle do teclado: chave, *dial* (rotativo) ou *slider* (deslizante).

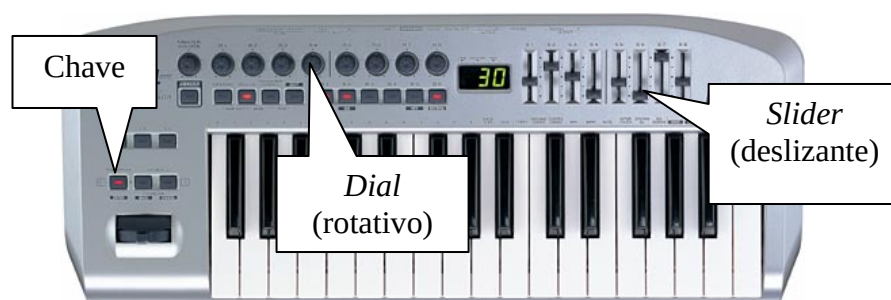


Figura 2.18 – Controles: Chave, Dial (rotativo) e Slider (deslizante)

¹⁴ Foi comprovado experimentalmente que existe uma relação linear do volume com que uma nota musical é produzida com a velocidade de acionamento da mesma, seja em tecla ou em corda. Assim, volume de uma nota, no protocolo MIDI é denominado por “*Velocity*” (velocidade).

¹⁵ Teclado sensível: aquele que possui um sensor de velocidade que permite registrar o volume com que cada nota é acionada, permitindo a transmissão fiel da dinâmica de execução do instrumentista no teclado controlador.

Assim, para ativar uma nota, até agora sabe-se necessitar de 2 Bytes. Ao Byte de nota e volume deu-se o nome de Byte de Dados. Falta agora definir quantos Bytes são necessários para representar a ação de ativar a nota para completar a mensagem de ativar nota. A sequência desta mensagem fica, até então:

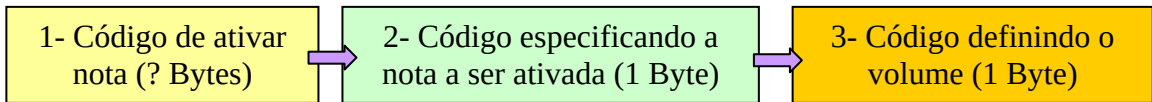


Figura 2.19 – Ilustração da sequência para ativar nota

2.6.1 Formato do comando de ativar notas.

Respondida a questão de como representar a nota ativada e seu respectivo volume, resta saber como é o formato do comando de ativar nota.

Para se entender este comando, deve-se antes, voltar ao tempo de quando o protocolo MIDI foi criado. Nesta época estava-se interessado em permitir que bandas musicais utilizassem ao mesmo tempo vários teclados controlados por um único controlador. Cada escravo seria ajustado com um timbre musical diferente, podendo ser ativado em solo ou em conjunto com um ou mais escravos. Esta escolha seria feita pelo teclado *master* (controlador). Assim, o comando de ativar nota deverá levar também a instrução de qual teclado deverá produzir o som da nota desejada.

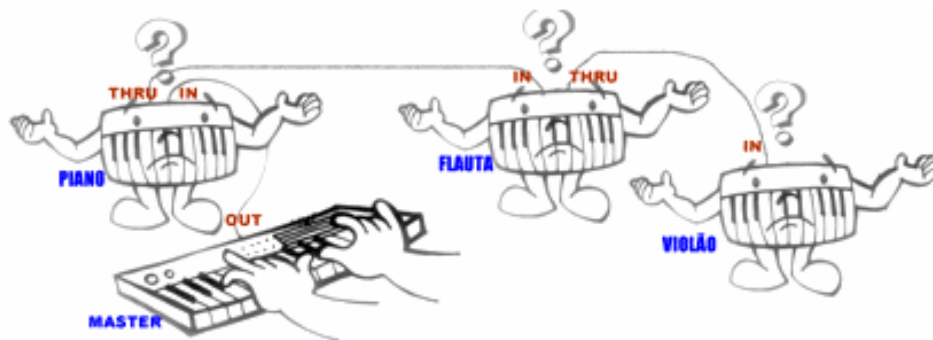


Figura 2.20 – Quem o controlador master controla?

Para que o teclado *master* possa ativar o teclado *slave* desejado, cada escravo deverá possuir um identificador, um código que o diferencie de outro escravo. Desta forma, o teclado controlador poderá enviar mensagens, como a de ativar notas, para qualquer teclado escravo dele.

O número de códigos dos escravos foi limitado pela unidade de trabalho do processador, o Byte. Os idealizadores chegaram à conclusão por utilizar apenas um Byte para enviar ao mesmo tempo o comando desejado e o respectivo código de qual escravo deverá executá-la. A este byte deu-se o nome de Byte de Status. Dividiu-se este Byte de Status em duas partes iguais (dois *nibbles*), cada um com 4 bits. Desta forma, com 4 bits permite-se gerar código para 16 escravos diferentes, cada um ativando um determinado timbre. A estes códigos deu-se o nome de CANAL MIDI.

À primeira parte do Byte dá-se o nome de MSB (a parte mais (*Most*) significativa do Byte) e à segunda parte, o nome de LSB (a parte menos (*Least*) significativa). Para diferenciar um Byte de Status de um Byte de Dados, utiliza-se o bit mais significativo do byte. Se o mesmo for 1, o Byte é de Status, se o mesmo for 0, o Byte é de Dados.

As figuras 2.21 e 2.22 mostram respectivamente um Byte de Status e um de Dados, com os respectivos *nibbles* de comando e de canal em uma mensagem de execução em tempo real.

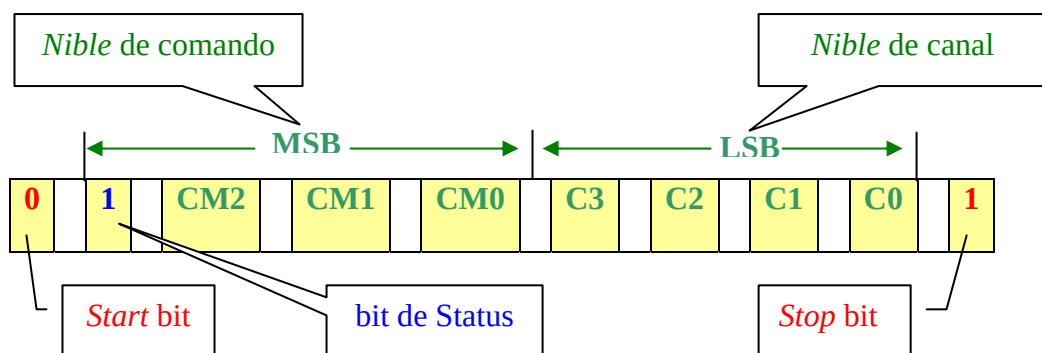


Figura 2.21 – Byte de Status

Onde: CM2, CM1 e CM0 representam os bits de comando e C3, C2, C1 e C0 representam os bits de canais.

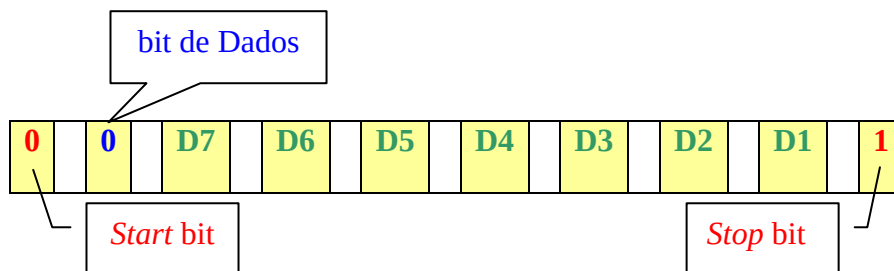


Figura 2.22 – Byte de Dados

Onde: D7, D6, D5, D4, D3, D2, D1 e D0 representam os bits de dados, no caso, de notas ou volume.

O protocolo MIDI define um código binário de três bits para os comandos que se referenciam às mensagens de canal¹⁶.

Assim o comando de ativar uma nota no canal 0 fica:

Dado:

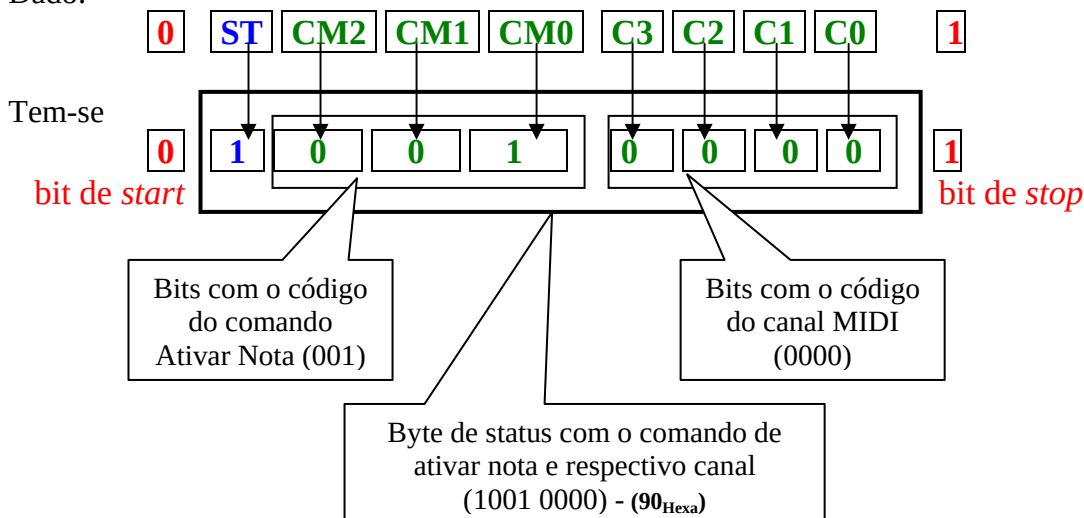


Figura 2.23 – Ilustração de um byte de status com o comando de ativar nota no canal MIDI 0

Desta forma, com três bits, pode-se codificar 8 tipos de comandos¹⁷ (3 bits $\rightarrow 2^3 = 8$). O Anexo IV mostra os tipos de mensagens de canal existente.

Como se utiliza um bit do byte para diferenciar um byte de status de um byte de dados, sobram 7 bits para codificação, gerando 128 códigos possíveis ($2^7 = 128$) para dados. Esta quantidade de códigos é suficiente para representar todas as notas do espectro audível (128 notas) e 128 volumes diferentes (mais do que o ouvido humano consegue discriminar). Assim, como exemplo, a nota lá diapasão de 440Hz possui o código 0011 1001 (39_{Hexa}).

Conclui-se, portanto, que para enviar uma mensagem de ativar uma nota musical em um determinado canal utiliza-se 3 (três) Bytes:

- um de status para informar o comando e o canal MIDI;
- outro de dados com a nota desejada;
- outro de dados com o volume da nota.

¹⁶ Uma mensagem de canal é aquela que é enviada para um canal MIDI específico, como, por exemplo, de ativar uma nota musical. Neste caso, deve-se informar em qual canal (instrumento) esta nota deverá ser ativada. No protocolo MIDI este código é 001, que, somado ao bit de status, fica 1001 (9_H), se o canal for o canal 0, o byte completo ficaria 1001 0000 (90_H)

¹⁷ Não confundir comando com mensagem. Uma mensagem pode possuir um comando, como no caso de ativar uma nota, seguida de mais dois bytes informando a nota e o respectivo volume.

Para exemplificar, os bytes para enviar a mensagem de ativar a nota lá 440Hz no canal 0 e volume máximo¹⁸(127), fica:

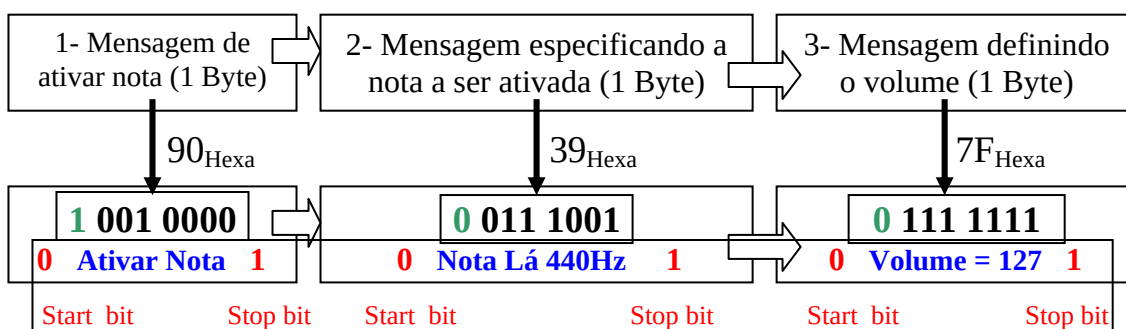


Figura 2.24 – Mensagem de ativar nota Lá 440Hz no canal MIDI 0

O que responde a pergunta inicial de como o protocolo MIDI representa a ação de ativar uma nota musical.

2.6.2 Mensagens MIDI e número de bytes por mensagens.

Foi visto que para se enviar uma mensagem no protocolo MIDI, a mesma utiliza Bytes de Status e Byte de Dados. Assim, dependendo o tipo de mensagem (controles, comandos, outras), o protocolo utilizará um ou mais Bytes.

Tomando como exemplo, o ato de enviar uma mensagem para ativar uma determinada nota musical utiliza três Bytes, a saber:

- **um Byte de status:** contendo o comando de ativar nota em um determinado canal
- **dois Bytes de dados:** um Byte para informar a nota a ser ativada e outro Byte pra informar o volume da mesma.

Uma mensagem MIDI sempre inicia por enviar um Byte de status para preparar a máquina MIDI para receber os dados e executar a ação que se deseja ver efetivada.

O que diferencia, portanto, o Byte de Status de um Byte de dados, é o bit mais significativo do mesmo: se o mesmo for 1, o Byte é de Status, se for 0, o Byte é de Dados.

¹⁸ Os valores vão de 0 a 127, totalizando 128 valores.

Quando o bit mais significativo de um Byte for 1, o valor do Byte sempre será maior que 127, já que o valor do bit mais significativo é 128.

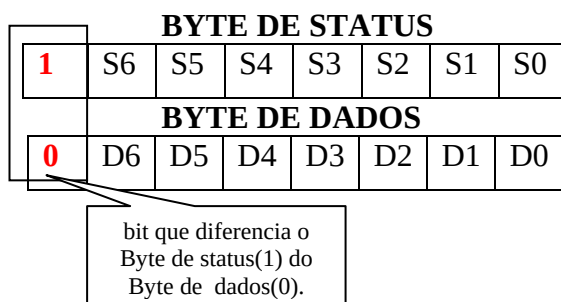


Figura 2.25 – Byte de status e Byte de dados

Onde:

- S0 a S6 são os 7 bits do Byte de Status
- D0 a D6 são os 7 bits do Byte de Dados

Assim, em uma mensagem MIDI utiliza-se apenas os **7 bits menos significativos em qualquer Byte de mensagens**¹⁹. Desta forma, quando uma máquina MIDI recebe uma mensagem cujo bit mais significativo for igual a 1, a mesma já se prepara para interpretá-la e realizar a ação desejada.

2.6.3 Ativando e desativando uma nota musical.

Sempre que uma nota musical for ativada, a mesma deverá ser desativada posteriormente²⁰. Assim, tocar uma nota musical por um determinado tempo ou figura musical, é um ato que demanda duas mensagens MIDI:

- Uma para ativar²¹ a nota musical desejada em um determinado canal MIDI
- Outra para desativar²² esta nota

Assim, ao se pressionar uma determinada tecla em um teclado musical MIDI (atualmente praticamente todos), o mesmo envia uma mensagem no protocolo MIDI de

¹⁹ Existem exceções que logo serão apresentadas ainda neste capítulo, tais como: mensagens de meta evento, mensagens exclusivas e outras.

²⁰ Caso contrário, se a nota não for desativada, a mesma ficará soando até que se cancele o processo da placa de som do computador, o que demanda, do usuário, conhecimentos técnicos menos triviais, ou, em alguns sistemas operacionais ou programas que não cessam seus processos quando desativados, de se ter até que desligar a máquina.

²¹ Ativar nota - > código 90_{Hexa} (144₁₀) ao código 9F_{Hexa} (159₁₀) -> ativar nota no canal 0 até ativar nota no canal 15.

²² Desativar nota-> código 80_{Hexa} (128₁₀) ao código 8F_{Hexa} (143₁₀) .

ativar nota para um escravo (módulo de som externo ou interno controlado pelo mesmo), e, ao liberar a tecla, outra mensagem é enviada ao escravo para desativá-la.

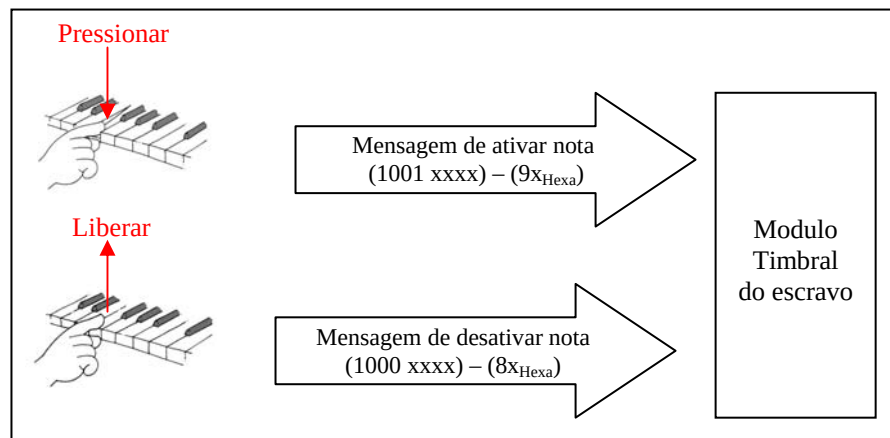


Figura 2.26 – Ativando e desativando notas

Onde x indica número do canal MIDI: 0000(0) -> canal 0 e 1111(F) -> canal 15, totalizando 16 canais.

O Anexo IV, Lima [2], Lopes [3] e Machado [1] mostram com mais detalhes a formalização e os tipos de mensagens MIDI existentes, com respectivo código de Status e o número de Bytes que cada uma possui²³.

Para dar mais um exemplo ilustrativo, observe uma mensagem de mudar o tipo de instrumento de um determinado canal (*Program Change*). Esta mensagem precisa de um código, do canal do instrumento e de qual o novo instrumento se deseja para este canal.

Da mesma forma que a mensagem de ativar ou desativar nota, com um Byte se consegue enviar o código da ação (mudar instrumento) e o canal desejado. Assim, tal mensagem demanda apenas dois Bytes: um Byte de Status (com o código do comando de mudar o instrumento com respectivo canal) e um Byte de Dados com o código MIDI do instrumento desejado.

²³ Cada mensagem demanda um número de Bytes diferente, conforme o que se deseja que o escravo faça.

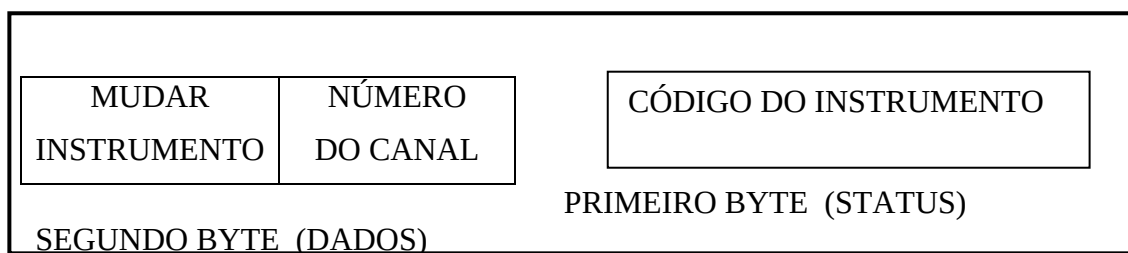


Figura 2.27 – Mensagem de mudança de instrumento – 2 bytes

O código de mudar o instrumento, incluído o bit de status, é 1011 ($C_{\text{Hexa}} - 12_{10}$). Assim, para mudar o instrumento do canal 0 (0000_{Hexa}) para violão (código 0001 1001 - $19_{\text{Hexa}} - 25_{10}$), a mensagem completa, com dois Bytes, fica:

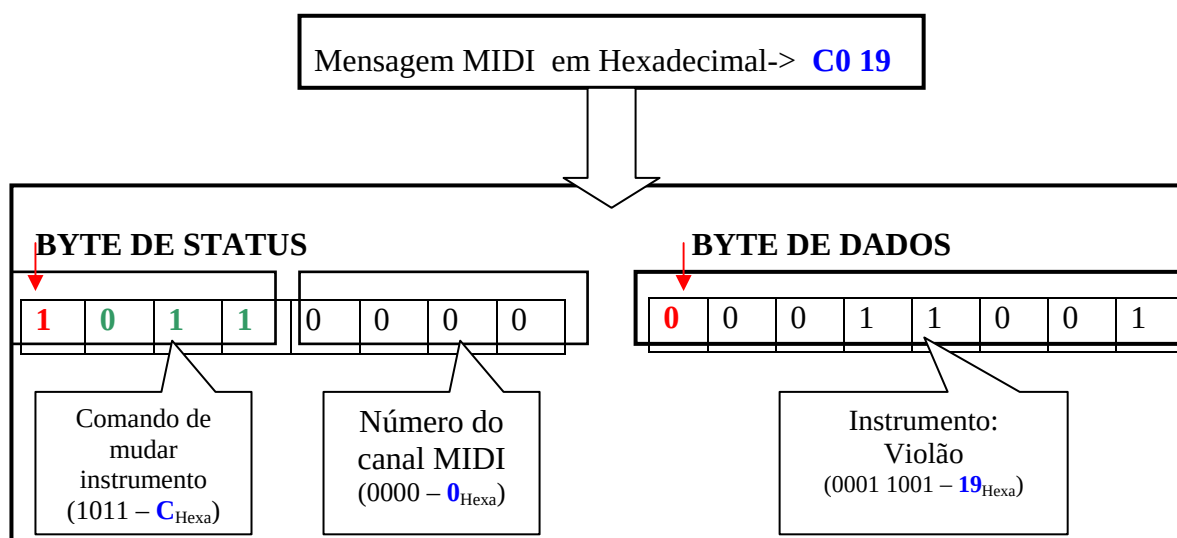


Figura 2.28 – Mudança de instrumento no canal 0

Estes tipos de mensagens, mensagens de canal, possuem número fixo de Bytes.

Por outro lado, existem mensagens que não possuem número fixo de Bytes, como, por exemplo, uma mensagem de texto.

Este tipo de mensagem exige um ou mais Bytes entre o Byte de Status e o(s) Byte(s) de dados.

Este(s) Byte(s) extra é(são) denominado(s) de **Contador**.

O(s) mesmo(s) serve(m) para informar quantos Bytes de dados a mensagem terá, o que pode ser de 0 Bytes a 268.435.455 Bytes, conforme será visto logo a seguir.

Existem dois tipos básicos de mensagens que necessitam de um contador de número de Bytes:

- as mensagens de **Meta Eventos**²⁴ ;
- as **Mensagens Exclusivas de sistema**²⁵ (SysEx).

As mensagens de Meta Eventos possuem o seguinte formato:

- Primeiro Byte: Byte de status: FF_{Hexa} (1111 1112 = 25510)
- Segundo Byte: Código de qual é o Meta Evento
- Contador com 1 a 4 Bytes contendo a informação de quantos Bytes de dados o Meta Evento terá
- Byte de dados do Meta Evento (se for de texto, seguem os caracteres, por exemplo).

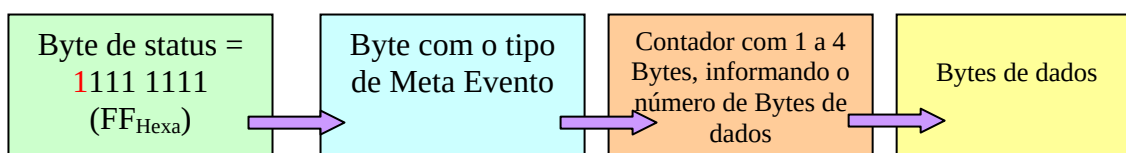


Figura 2.29 – Mensagem de Meta Eventos

A Mensagem exclusiva também segue a mesma estrutura dos Meta Eventos.

2.6.4 Byte de dados com 8 bits: uma exceção à regra geral.

Como foi dito, a máquina MIDI fica sempre analisando o bit mais significativo de um Byte para ver se o mesmo é um Byte de status. Após este Byte de status a máquina fica esperando os Bytes de dados para completar o comando.

Foi dito que os Bytes de dados possuem o bit mais significativo igual a 0. Esta é a regra geral que, como toda regra, possui exceções (o clássico paradoxo).

A exceção, no caso, são as mensagens exclusivas de sistemas, mensagens de tempo real e as mensagens de meta Eventos. Nestes tipos de mensagens, cujo formato foi apresentado anteriormente, a máquina MIDI não precisa ficar checando o tempo todo o bit mais significativo de cada Byte para saber se o Byte corrente é um status ou dados.

²⁴ Um meta evento é um evento que leva informações relevantes à música, tais como: tempo (metrônomo), fórmula de compasso, tonalidade, lirismo, texto, copyright, outros.

²⁵ Uma mensagem exclusiva é uma mensagem que leva informações exclusivamente para um determinado equipamento de um fabricante. Ela é necessária devido cada teclado possuir recursos adicionais diferentes dos outros, e, caso se queira utilizar estas potencialidades extras, deve-se mandar mensagens que sejam exclusivas aos mesmos. Cada fabricante tem um código cadastrado e conhecido pelos fabricantes de teclado e equipamentos MIDI.

Estas mensagens possuem um contador de número de Byte de dados, e, desta forma, todos os Bytes após o(s) Byte(s) de contagem são Bytes de dados.

Esta característica permite que se possa utilizar os 8 (oito) bits dos Bytes como bit de dados. Isto se fez e faz necessário para que se possa, por exemplo, enviar mensagens de textos, lirismo e outras mais, as quais necessitam de 8 bits para representar, em ASCII²⁶, todos os caracteres necessários a um texto.

Existem outros padrões mais completos para representar diversas áreas do conhecimento, tal como o padrão UNICODE²⁷, o qual independe de idioma, programa ou plataforma.

2.6.5 O Contador MIDI de 1 a 4 Bytes.

Entender como é formada uma contagem no protocolo MIDI, com um ou mais Bytes, é de fundamental importância tanto para se poder implementar programas que interpretem corretamente mensagens do tipo SysEx ou de Meta Eventos, que utilizam os contadores para registrar o número de Bytes de dados das mensagens, quanto, posteriormente, para se poder ler um arquivo MIDI, onde os contadores possuem mais uma função: registrar o tempo de duração de uma nota²⁸ ou para registrar, cronologicamente, a ocorrência de todos os Eventos MIDI ocorridos durante a execução da música.

Converter 4 Bytes para um valor inteiro equivalente é um processo relativamente simples. Basicamente é converter o valor de cada Byte para o inteiro equivalente, multiplicá-los pelo peso de cada Byte e somar os valores resultantes, como segue:

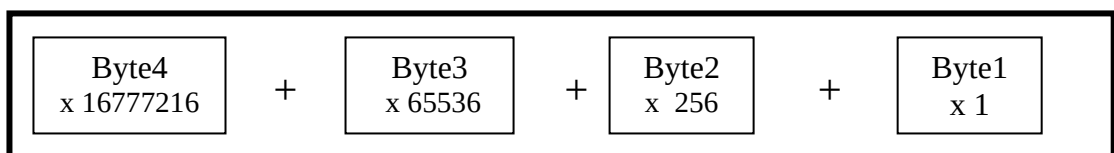


Figura 2.30 – Conversão de 4 Bytes para inteiro

²⁶ ASCII – **American Standard Code for Information Interchange** (<http://pt.wikipedia.org/wiki/ASCII>)

²⁷ UNICODE - O **Unicode** abrange quase todas as escritas em uso atualmente, além das escritas históricas já extintas e símbolos, em especial, os matemáticos e os musicais. (<http://pt.wikipedia.org/wiki/Unicode>)

²⁸ Este contador registrará o tempo decorrido entre a ação de pressionar uma nota e a ação de liberá-la. Registrando este tempo, poder-se-á reproduzi-la posteriormente com fidelidade.

Os valores dos multiplicadores são calculados pela potência do bit menos significativo de cada Byte, ou seja:

Primeiro Byte -> bit menos significativo = multiplicador = $2^0 = 1$

Segundo Byte -> bit menos significativo = multiplicador = $2^8 = 256$

Terceiro Byte -> bit menos significativo = multiplicador = $2^{16} = 65.536$

Quarto Byte -> bit menos significativo = multiplicador = $2^{24} = 16.777.216$

Assim, observe o exemplo de se transformar 4 Bytes em binário para um inteiro equivalente, cujos valores de cada um dos bytes são:

- Byte4 = 200;
- Byte3 = 120;
- Byte2 = 40;
- Byte1 = 10;

O valor em inteiro ficaria:

$$\text{Valor total} = (\text{valor do Byte4}) + (\text{valor do Byte4}) + (\text{valor do Byte4}) + (\text{valor do Byte4}) =$$

$$(200 \times 16.777.216) + (120 \times 65.536) + (40 \times 256) + (10)$$

-> Valor total = 3.363.317.770

Como se pode perceber, converter Bytes para um valor inteiro equivalente é uma tarefa simples. Infelizmente, converter um ou mais Bytes de um contador MIDI, para inteiro, **não segue esta mesma regra**, já que em um contador MIDI não se utiliza os oito bits do byte (o oitavo bit, cujo valor é $128(2^7)$).

Assim, se um contador possuir contagem maior que 127^{29} , isto significa que o bit mais significativo do Byte, o oitavo, é igual a 1. Neste caso, se tiver que saber o real valor da contagem deste Byte, deve-se subtrair 128 do mesmo

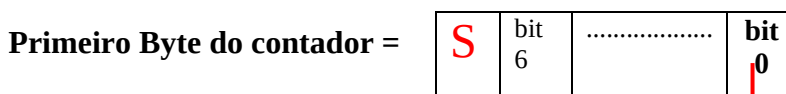
CONVERTER UM BYTE BINÁRIO PARA INTEIRO É DIFERENTE DE CONVERTER UM BYTE DE CONTAGEM MIDI PARA INTEIRO.

²⁹ O valor 127 equivale ao Byte 0111 1111. Valores superiores a 127 o byte terá o bit 7 (o oitavo bit) igual a 1. observe que $128 = 1000\ 0000$. Assim, em um contador com contagem 1000 0000 ou 0000 0000, equivalem ao valor inteiro 0. Os bits em vermelho, os mais significativos, apenas indicam status. Assim, se subtrair do byte 1000 000 o valor 128, tem-se 0, que é o valor real da contagem.

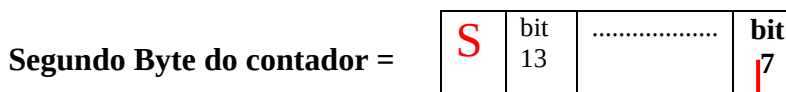
O problema de converter uma contagem MIDI para inteiro é que os pesos (multiplicadores para cada Byte), não são os mesmos para os contadores MIDI, já que só se utilizam 7 bits em cada Byte do contador MIDI.

Assim, para converter uma contagem MIDI para inteiro deve-se ajustar os multiplicadores de cada Byte e eliminar a contribuição do oitavo bit (bit de Status/Dados) da conversão de cada Byte.

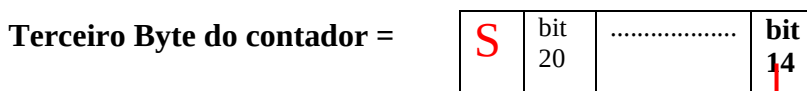
Para tanto, os pesos de cada byte, em um contador MIDI, ficam:



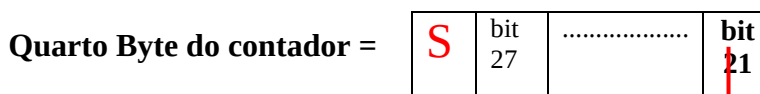
Primeiro Byte -> bit menos significativo = bit 0 -> multiplicador do Byte = $2^0 = 1$



Segundo Byte -> bit menos significativo = bit 7 -> multiplicador do Byte = $2^7 = 128$



Terceiro Byte -> bit menos significativo = bit 14 -> multiplicador do Byte = $2^{14} = 16384$



Quarto Byte -> bit menos significativo = bit 21 -> multiplicador do Byte = $2^{21} = 2097152$

Onde **S** = bit que indica o Status do Byte, o qual não é levado em conta na conversão da contagem.

Assim, se o oitavo bit de um Byte do contador, bit **S**, for igual a 1, deve-se subtrair do valor do Byte o valor 128 (que é o valor da potência do oitavo bit: $2^7 = 128$). Se o mesmo for 0, a contagem é o próprio valor do Byte.

2.6.5.1 Regra para se determinar o número de bytes de um contador MIDI.

A regra para se determinar quantos Bytes um contador MIDI tem é a seguinte.

- Se o Byte possuir valor maior que 127, bit **S** = 1, isto indica que o contador terá mais um Byte e que se deve subtrair 128 do valor deste byte na conversão para inteiro;
- Se o Byte possuir valor menor ou igual a 127, bit **S** = 0, isto indica que os Bytes de contagem terminaram. Como o bit mais significativo deste Byte é 0 (bit **S** = 0), não é preciso, portanto, subtrair nenhum valor deste Byte na conversão para inteiro.

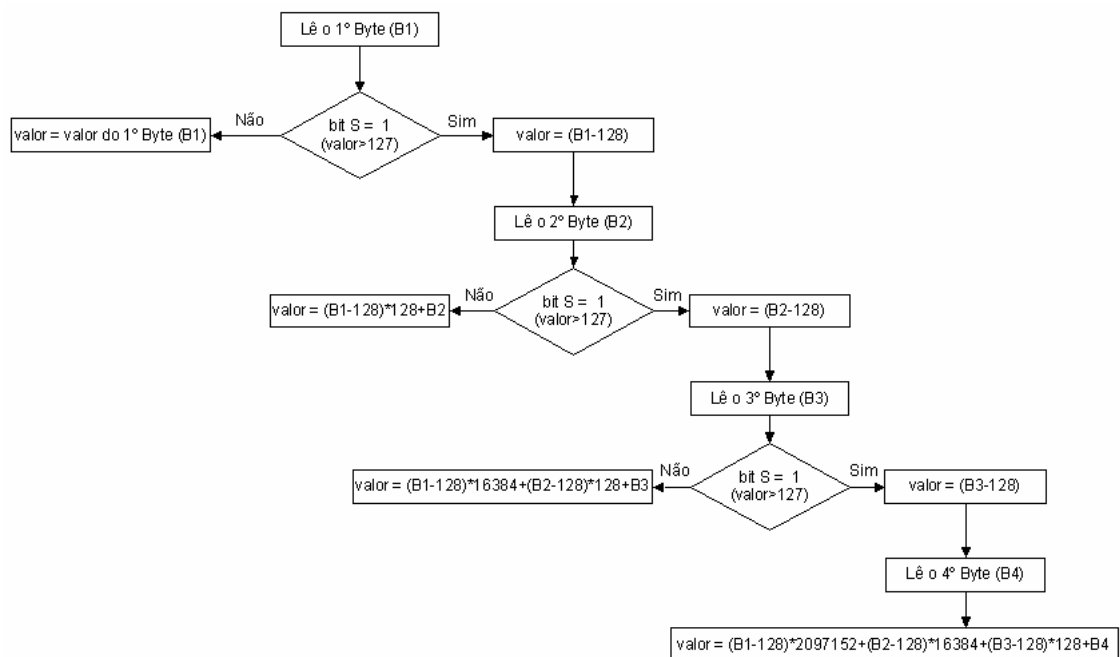


Figura 2.31 – Fluxograma: Conversão de um Contador MIDI para inteiro.

2.6.5.2 Fórmulas para se converter uma contagem de 1 a 4 Bytes em um inteiro equivalente.

Partindo dos conceitos apresentados, pode-se equacionar a conversão de contagem MIDI para um valor inteiro equivalente, partindo do valor em inteiro de cada Byte da contagem, sem que se precise trabalhar com os bits de cada Byte separadamente.

Número de Bytes	Calculo da conversão da contagem em valor inteiro
1 Byte = B0	B0
2 Bytes = B1 B0	$((\mathbf{B1} - 128) \times 128) + \mathbf{B0}$
3 Bytes = B2 B1 B0	$((\mathbf{B2} - 128) \times 16384) + ((\mathbf{B1} - 128) \times 128) + \mathbf{B0}$
4 Bytes = B3 B2 B1 B0	$((\mathbf{B3} - 128) \times 2097152) + ((\mathbf{B2} - 128) \times 16384) + ((\mathbf{B1} - 128) \times 128) + \mathbf{B0}$

Tabela 2.1 – Conversão da contagem de 1 a 4 Bytes em valor inteiro

No Anexo V é mostrado com mais detalhes os valores máximos e mínimos de cada contagem com 1 Byte, 2 Bytes, 3 Bytes e 4 Bytes.

2.7 Classificação das mensagens quanto ao número de bytes.

Para que se possa capturar, receber corretamente uma mensagem MIDI, basta saber quantos Bytes a mesma possui. Uma vez separada a mensagem pode-se, através de seu(s) Byte(s) de Status, identificá-la.

Quando o protocolo MIDI foi criado, uma regra básica foi estabelecida a todos os fabricantes e desenvolvedores de equipamentos MIDI (software e Hardware): Se uma mensagem não for reconhecida ou passível de ser executada pelo mesmo, a mesma deverá ser desconsiderada.

Assim, para desconsiderar uma mensagem que não consegue executar ou interpretar corretamente, o sistema deverá pelo menos saber quantos Bytes a mesma possui para poder aguardar a próxima mensagem.

Como o objetivo principal deste trabalho é permitir que se possa facilitar o desenvolvimento de novos aplicativos MIDI, é de fundamental importância, portanto,

que se saiba como separar uma mensagem MIDI recebida (mesmo que não a compreenda).

A seguir, é mostrado como separar estas mensagens conhecendo o código de seu Byte de status e seu respectivo número de Bytes.

Nº de Bytes	Código de Status
1 Byte	241,246,247,(248,250,251,252,254,255->tempo real)
2 Bytes	192 a 223 e 243
3 Bytes	128 a 191 e 224 a 239 e 176 a 191 e 242
n Bytes	255,240

Tabela 2.2 – Mensagens MIDI por nº de bytes

2.8 Resumindo o envio de mensagens em tempo real.

Quando uma tecla de um controlador MIDI é pressionada, ocorre o seguinte:

- O controlador identifica a tecla pressionada e a velocidade com que a mesma foi acionada (a velocidade é proporcional ao volume com que a mesma deverá ser reproduzida)
- O controlador gera uma mensagem MIDI e a transmite ao(s) equipamento(s) escravo(s).

Na transmissão da mensagem, ocorre o seguinte:

1. O teclado controlador transmite uma mensagem de status de ativar nota em um determinado canal³⁰.
2. Esta mensagem chega ao Port MIDI IN do módulo de som (do teclado, placa de som, módulo timbral,...) e é, então, decodificada pelo processador do equipamento MIDI. Este equipamento, tendo um módulo de som, emite, faz soar a nota com um determinado volume. A nota musical e o volume são enviadas em 2 (dois) bytes de dados (mensagem com 3 Bytes) que seguem o Byte de status..

³⁰ Normalmente o canal em que a mensagem é transmitida é o canal 0 (zero) por default

3. Após a emissão desta mensagem, o equipamento MIDI escravo fica aguardando a próxima mensagem, que pode ser uma nova ativação de nota, uma desativação de nota, uma mudança de instrumento, ou uma outra mensagem qualquer.
4. Se o próximo Byte for de status, valor maior que 127, o sistema armazena o mesmo, define o número de Bytes de dados que comporão a mensagem, e, posteriormente, o equipamento escravo executa a ação da mensagem.
5. Caso o próximo Byte não seja de Status (bit mais significativo igual a 0), e sim de dados, o sistema adota o status anterior (running status) e aguarda a conclusão do envio dos bytes de dados para novamente efetuar a ação.
6. Caso se queira ativar uma nova nota, após uma mensagem anterior de ativar nota, pode-se utilizar o status da mesma, running status, e somente enviar o Byte de Dados com o código da nota e o Byte de dados com o volume.
7. Caso se queira desativar uma das notas ativadas, basta enviar o Byte de Status (80) indicando que se deseja desativar a nota, seguido do Byte de dados contendo o código da nota a ser desativada e o volume com código 0.
8. Caso desejado for, para simplificar a emissão de Bytes de status, ao se desativar uma nota, basta enviar o Byte de dados contendo a nota desejada e o Byte de volume com valor 0, aproveitando o running status (status corrente).

É importante destacar que quando o teclado é ligado, o mesmo carrega todos os seus 16 canais com os respectivos instrumentos e controles padrão, enviando os mesmos ao seu módulo de som ou a seu(s) equipamento(s) escravo(s).

2.9 Arquivos MIDI padrão: Os **Standard MIDI Files (SMF)**.

O protocolo MIDI, além de permitir a interligação entre teclados e equipamentos MIDI para apresentações ao vivo, em tempo real, também especifica formatos³¹ de arquivos MIDI para o armazenamento e posterior reprodução com fidelidade das seqüências musicais. A estes tipos de arquivos, deu-se o nome de **SMF: Standard MIDI Files**.

2.9.1 A máquina MIDI.

Para armazenar e executar os arquivos SMF, foi criada uma máquina MIDI com uma arquitetura bastante simples e que pudesse ser implementada a baixo custo e com a tecnologia limitada da época de sua concepção. Assim, a mesma deveria possuir um número pequeno de componentes que pudessem executar todos os comandos necessários para se registrar e reproduzir uma seqüência musical, seja por hardware ou por software.

Este objetivo resultou em uma máquina que possui apenas um contador, um registrador/decodificador de eventos (status) e um registrador de dados. Ou seja, uma máquina que consegue armazenar uma mensagem MIDI, decodificá-la e saber quando deverá executá-la.

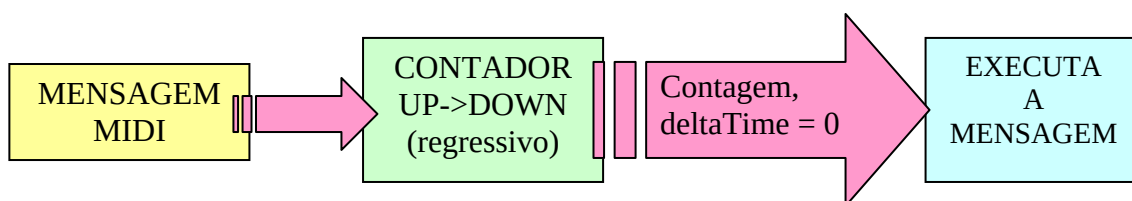


Figura 2.32 – Máquina MIDI

³¹ Formato- É a forma, a estrutura, com que as informações serão registradas em arquivo. Não confundir com protocolo. Nos arquivos SMF, tem-se os formatos 0 e 1 onde as informações (eventos) MIDI serão registradas seguindo um protocolo. A estrutura de como as informações neste protocolo serão armazenadas é denominada de formato.

2.9.2 Como a máquina MIDI grava e lê mensagens em SMF.

2.9.2.1 Gravando mensagens em Standard MIDI Files.

Para entender como uma mensagem MIDI é armazenada utilizando o protocolo MIDI em arquivos SMF, é interessante recordar como os eventos MIDI são executados em apresentações ao vivo, em tempo real.

Em tempo real, a cada ação realizada pelo teclado mestre (controlador), corresponde uma reação no dispositivo escravo. Em outras palavras, por exemplo, se alguém aperta uma tecla do controlador, uma mensagem é enviada a um módulo de som (o equipamento escravo) que executa a nota musical correspondente, fazendo-a soar até que este alguém retire o dedo da tecla. Assim, o som da nota ativada dura enquanto a tecla da nota musical estiver pressionada, ou seja, enquanto o escravo não receber a mensagem de desativar a nota.

Observe que, nesta ação, duas mensagens MIDI foram enviadas:

- Uma de ativar a nota pressionada com respectivo volume;
- Outra com a mensagem de desativar a nota.

Observe, também, que apenas gravar o evento em mensagens de ativar e desativar uma nota musical, não garante a um equipamento MIDI, que as fossem ler, uma execução igual à original. Isto se daria devido ao fato de que o equipamento MIDI só teria acesso a estas duas mensagens, faltando um dado importante, ilustrado pelo seguinte questionamento:

- Quanto tempo a nota musical deverá ficar ativada?

O problema é que nas mensagens de ativar e desativar eventos não se tem o registro do tempo decorrido entre estas duas ações. Este tempo é que permitiria a um equipamento reproduzir com fidelidade as ações originais.

Para solucionar este problema, nos teclados e seqüenciadores que possuem o recurso de armazenar, gravar, as seqüências musicais (nem todos possuem), foi introduzido pelo padrão MIDI, um contador de tempo de duração de eventos MIDI.



Figura 2.33 – Teclado com recurso de armazenar seqüências musicais

Assim, sempre que uma ação é ativada, que um evento MIDI ocorre, um contador é disparado armazenando uma contagem equivalente ao tempo entre dois eventos consecutivos.

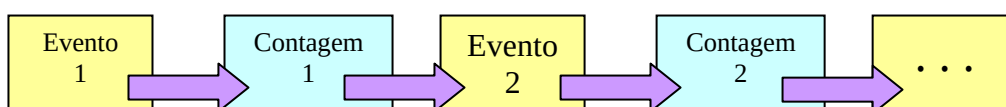


Figura 2.34 – Eventos e contagens

Nem sempre o tempo marcado pelo contador corresponde ao tempo em que um evento ficou ativo. Na ação entre ativar e desativar uma **mesma** nota musical isto ocorre, já que, quando o evento de desativar a nota é disparado, o som desta nota é finalizado.

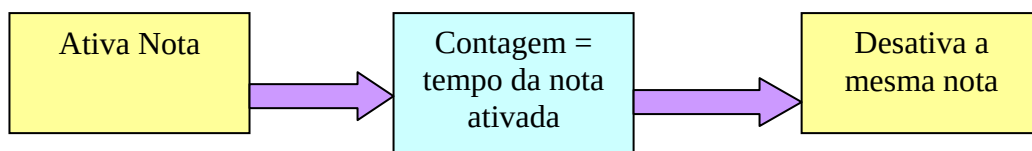


Figura 2.35 – Contagem = tempo de um evento ativo

Imagine agora o ato de ativar duas notas musicais em seqüência e desativá-las posteriormente. Neste caso ativa-se uma das notas e dispara-se o contador, o qual contará até que um novo evento ocorra, o que, neste caso, não será de desativar uma nota e sim de ativar a segunda nota. A contagem registrada, neste caso, não é a de duração do tempo em que a primeira nota musical, e sim do tempo decorrido entre a ação de ativar a primeira nota e a segunda nota.

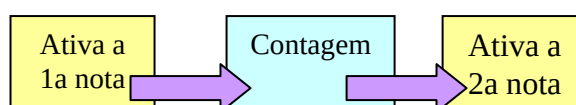


Figura 2.36 – Ativando uma segunda nota sem desativar a primeira.

Ativadas as duas notas, resta decidir qual delas desativar primeiro. Isto dependerá de quem está tocando as mesmas. Assim, pode ser que a primeira nota ficará soando mais tempo que a segunda ou que a segunda ficará soando por mais tempo que a primeira. Para entender como calcular o tempo de duração de cada nota, a seguir são analisados os tempos para as duas opções.

-Caso 1: Desativação da primeira nota antes da segunda nota

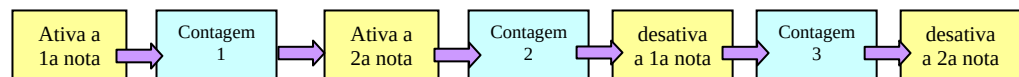


Figura 2.37 – ativando duas notas em seqüência

Assim, para se ter o tempo do evento de ativar a primeira nota, deve-se somar todos as contagens intermediárias entre o ato de ativar esta nota e o ato de desativá-la, ou seja:

-Tempo da nota 1 = Contagem 1 + Contagem 2

-Tempo da nota 2 = Contagem 2 + Contagem 3

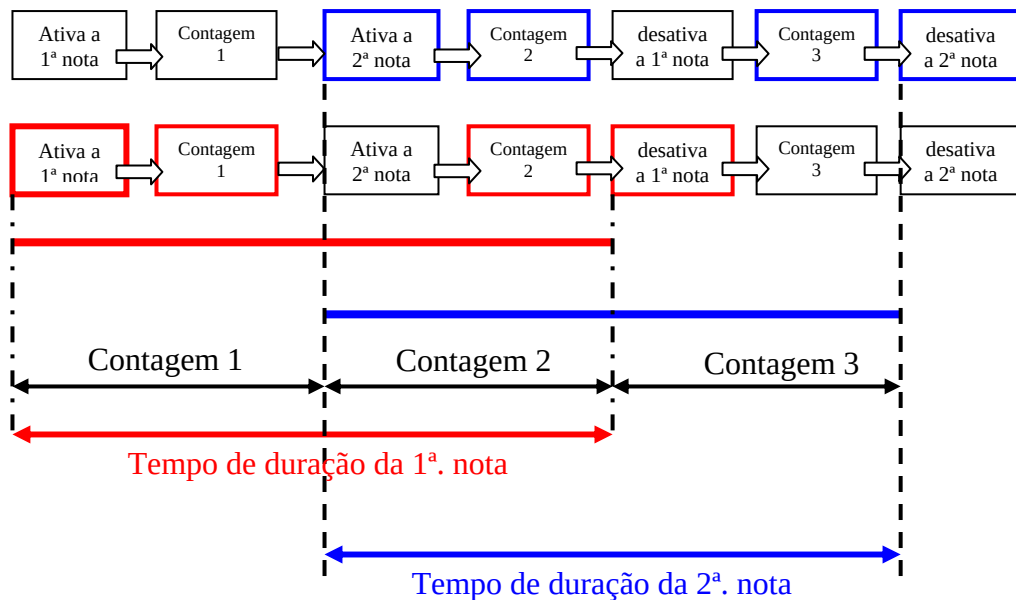


Figura 2.38 – Duração da 1ª nota e duração da 2ª nota

-Caso 2: Desativação da primeira nota depois da segunda nota

-Tempo da nota 1 = Contagem 1 + Contagem 2 + Contagem 3

-Tempo da nota 2 = Contagem 2

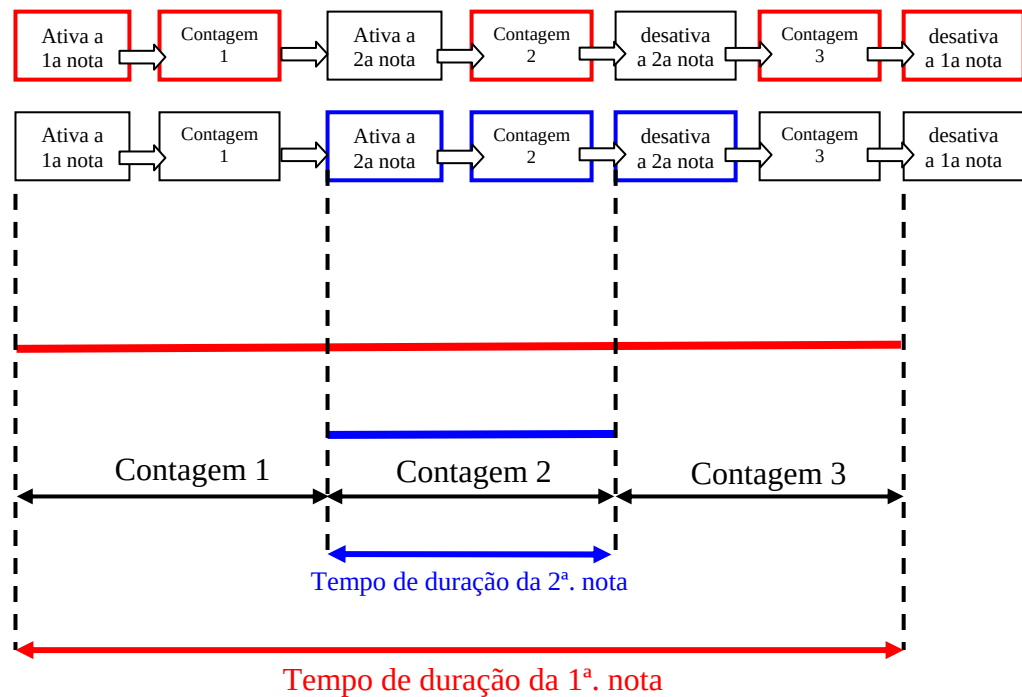


Figura 2.39 – Desativação da 1ª nota depois da 2ª nota

Neste caso, o tempo de duração do evento de ativar a segunda nota coincide com o tempo marcado pelo contador entre o evento de ativar e desativar nota.

Pode-se concluir, portanto, que não existe no protocolo MIDI a informação explícita do tempo de duração de um evento, ou seja, não se tem registrado explicitamente no mesmo uma informação tão simples como **ativar uma nota qualquer, com um determinado volume por um determinado tempo**. O fator tempo deverá ser inferido pelo equipamento que for ler o arquivo MIDI e executá-lo.

2.9.2.2 Ativando e desativando notas em um SMF.

Ativar uma nota e desativá-la posteriormente, para armazenamento em um SMF, é diferente do processo em execuções em tempo real. Observe a seqüência de ativar e desativar nota em uma máquina MIDI para gerar um SMF:

1. Armazena a mensagem de ativar uma nota musical em um determinado canal e seu volume;
2. Dispara o contador que acumulará uma contagem até que ocorra um novo evento que gere uma nova mensagem;
3. Ao receber a nova mensagem, a contagem é finalizada e armazenada³²;
4. Feito isto o contador é zerado para armazenar o tempo da nova mensagem³³;
5. Armazena a nova mensagem (desativar a nota musical) e repete-se o processo do item 2 ao 5 até que os eventos MIDI terminem e se finalize a geração do SMF.

A figura 2.40 mostra a seqüência de eventos de Ativar e Desativar nota com o registro do tempo em que a nota ficou ativada.



Figura 2.40 – Máquina MIDI: Ativa e Desativa Nota com Registro de tempo

³² A contagem é armazenada em um ou mais Bytes (de 1 a 4), conforme o tempo de duração do evento.

³³ Mesmo que se consiga ativar dois eventos ao mesmo tempo, como o de apertar duas notas ao mesmo tempo, o sistema guarda uma mensagem de ativar a primeira nota, logo após coloca um contador com contagem 0 e guarda a mensagem de ativar uma outra nota. Ou seja, depois de cada evento, é obrigatório se enviar um contador informando o tempo em que o evento ficou ativado até que um novo evento ocorreu.

2.9.2.3 Delta Time.

Dá-se o nome de **Delta Time** à contagem do tempo em que um evento MIDI ficou ativado.

Um delta time, é uma contagem registrada com 1(um) a 4(quatro) Bytes, ou seja, é um contador MIDI conforme já descrito anteriormente para se armazenar contagens de número de Bytes de Dados em uma mensagem MIDI do tipo Meta Evento, SysEx e outras.

2.9.2.4 Contagem MIDI x tempo cronológico.

Contar Bytes de dados com um contador é um processo fácil de entender. Observe:

-Se for desejado armazenar a informação de se ter 30 Bytes de dados, basta registrar no contador o valor 30. Esta informação, por si só já é completa, ou seja, a contagem 30 significaria 30 bytes.

Mas, em uma contagem de tempo, este mesmo valor 30 significaria o que?

- Que o tempo do evento é de 30 segundos?
- Que o tempo do evento durará o tempo de 30 semínimas?

Na realidade, não significa nem um nem outro, e, desta forma, para se responder a esta questão um novo conceito deve ser compreendido e assimilado. A contagem de tempo em um contador é uma contagem relativa de tempo, relativa ao tempo de duração da unidade padrão de tempo adotada pelo protocolo MIDI, relativa à contagem adotada para o tempo de uma semínima. Para ficar mais claro a explicação, observe como a contagem de tempo é realizada em uma partitura musical **CPN** (Common Practice Notation – uma partitura tradicional).

2.9.2.5 Tempo de uma nota musical em uma partitura.

Em um registro de música, como, por exemplo, uma partitura, o tempo de cada nota não é registrado em valores absolutos e sim em valores relativos (figuras musicais³⁴).

³⁴ Figuras musicais, do tipo: semibreve (whole note), mínima (half note), semínima (quarter note), colcheia (eighth note), semicolcheia (16th note), fusa (32th note), ..

Quando o músico for interpretar a música, o mesmo deverá ter a informação de como transformar as figuras musicais em tempo absoluto (segundos). Para tanto, em primeiro lugar o mesmo deve conhecer a relação temporal relativa entre as figuras musicais, como exemplificado a seguir:

- uma semibreve (whole note) dura quatro vezes (400%) do tempo de uma **semínima**;
- uma mínima (half note) dura o dobro (200%) do tempo de uma **semínima**;
- uma colcheia (eighth note) dura a metade (50%) do tempo de uma **semínima**.

Para se saber o tempo em segundos, de uma nota, partindo deste valor percentual denotado pelas figuras musicais, é notado em uma CPN um parâmetro que informa qual é a figura musical adotada como padrão de tempo³⁵. Este parâmetro informa quantas figuras padrão compõem um minuto (número de figuras por minuto). Este parâmetro, é denominado de Metrônomo. A figura 2.41 mostra uma CPN onde é mostrado o valor do metrônomo adotada nela. O valor 120 significa que o tempo básico da CPN é de 120 semínimas por minuto.



Do exposto, conclui-se que o metrônomo não informa diretamente o tempo de duração da figura musical (no caso a semínima), mas sim quantas unidades da mesma cabem em um minuto.

Para saber o tempo de uma semínima, em segundos, conhecendo o valor do metrônomo, basta fazer o seguinte cálculo:

$$\text{TempoSemínima} = (60/\text{Metrônomo}) \text{ segundos}$$

³⁵ A figura padrão de uma partitura é aquela que define a unidade tempo da música. Se a fórmula de compasso for 4:4, a unidade tempo (o denominador da fração) é a semínima (4), se é 4:2, a unidade é a mínima (2), se é 4:8, a unidade é a colcheia e assim por diante. No padrão MIDI, a unidade de tempo, independentemente da fórmula de compasso, será a semínima (quarter note).

ou seja:

Se o metrônomo possui valor igual a 120, isto significa que se tem 120 semínimas em cada 60 segundos(1 minuto), ou seja, que em cada segundo tem-se 2 semínimas. Portanto, o tempo de uma semínima será a metade do tempo de um segundo (0,5 segundos).

$$\text{-TempoSemínima} = 60/120 = 0,5 \text{ segundos.}$$

Desta forma, conhecido o valor do metrônomo, pode-se conhecer a duração temporal de uma semínima, e conhecida a duração da mesma, pode-se calcular o tempo das demais figuras musicais em valor absoluto.

Neste caso, se uma semínima dura 0,5 segundos, tem-se:

- uma colcheia = 50% (1/2) de uma semínima = 0,25 segundos;
- uma mínima = 200% (2x) de uma semínima = 1 segundo.

Quem definirá, portanto, o tempo em segundos de duração de uma nota musical, é uma associação de sua figura musical com o valor do metrônomo.

2.9.2.6 Tempo de um evento em um arquivo SMF.

Foi visto como calcular o tempo de uma figura musical em uma CPN.

Em um arquivo MIDI SMF, apesar do cálculo ser feito de forma diferente (logo será mostrado como), o tempo registrado por um contador (o qual marca a duração de um evento) também é um valor relativo.

A diferença da CPN com um SMF é que para se calcular a duração de uma nota musical não basta conhecer o valor registrado por um contador. O tempo de duração de uma nota não é explicitado por um determinado contador. O mesmo depende do contexto, ou seja, depende de se saber se entre ativar uma nota e desativá-la não ocorreram outros eventos intermediários. Caso ocorra, o tempo de duração da nota será o resultante de todas as contagens (Delta Times) existentes entre o ato de ativar a nota e desativá-la.



Figura 2.42 – Tempo relativo da nota

Caso não exista nenhuma mensagem entre o ato de ativar e desativar uma nota, a contagem (o Delta Time) existente entre estes dois eventos será o tempo relativo da nota musical em questão.

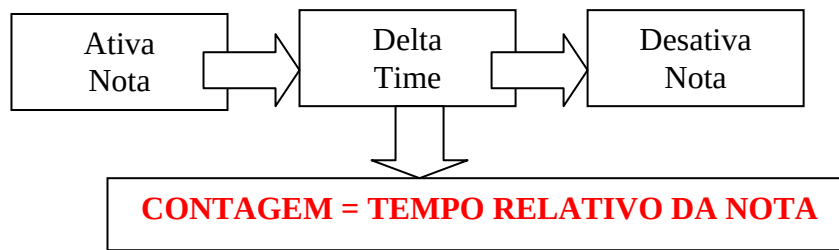


Figura 2.43 – Contagem – tempo relativo da nota

Foi apresentado que a contagem de tempo entre eventos, os Delta Times, são contagens de tempo relativas. Isto se dá porque as contagens são obtidas por computadores e outros equipamentos MIDI (teclados, seqüenciadores, etc.) com velocidades de processamento diferentes³⁶. Neste caso, os intervalos de tempo entre contagens e as contagens propriamente ditas possuirão um valor temporal diferente.

Para exemplificar e tornar mais claro o que foi dito, observe o seguinte exemplo:

-Se um computador possui um relógio (*clock*) de 1.8GHz, isto significa que o mesmo pode realizar até 1.800.000.000 contagens por segundo. Se outro computador

³⁶ A velocidade de processamento de uma máquina depende, fundamentalmente, de um dispositivo interno que marca o tempo entre instruções do processador. Este dispositivo é chamado de relógio ou de *clock*. Quando o protocolo MIDI foi criado, os computadores pessoais possuíam *clocks* com algumas dezenas ou centenas de MHz (1.000.000 de Hz), e, atualmente, com valores que superam alguns GHz (1.000.000.000 de Hz).

possuir um *clock* de 300 MHz, o mesmo realizaria apenas 300.000.000 contagens por segundo.

Destes dados, suponha obter, com estes dois computadores, uma contagem igual ao valor 300.000. Assim, será que estas contagens, realizadas por duas máquinas com velocidades diferentes, representam o mesmo valor temporal (em segundos)? Para tanto, observe as explicações e cálculos que seguem:

1-

Contagem realizada pelo computador de 1.8GHz

1.800.000.000	->	1 segundo
300.000	->	X

Onde X é o tempo que se deseja conhecer:

Aplicando a regra de três simples tem-se:

$$X = 300.000 / 1.800.000.000 = \mathbf{0,000167 \text{ segundos}}$$

$X = \mathbf{0,000167 \text{ segundos}}$
--

2-

Contagem realizada pelo computador de 300MHz

300.000.000	->	1 segundo
300.000	->	X

Onde X é o tempo que se deseja conhecer:

Aplicando a regra de três simples tem-se:

$$X = 300.000 / 300.000.000 = \mathbf{0,001 \text{ segundos}}$$

$X = \mathbf{0,001 \text{ segundos}}$

Pode-se observar que existem duas contagens com valores iguais (300.000), as quais, registradas por máquinas diferentes, significam tempos diferentes, ou seja:

- No primeiro computador, o de 1.8GHz, a contagem 300 equivale a 0,00016 segundos;

- No segundo computador, o de 300MHz, a contagem 300 equivale a 0,001 segundos (6 vezes maior).

Assim, o padrão MIDI definiu uma forma para possibilitar que estas contagens de tempo relativas possam ser unificadas em contagens de tempo absolutas, independentemente da velocidade da máquina que efetuou a contagem.

Para se obter o tempo em segundos a partir de um Delta Time (a contagem), dois fatores devem ser levados em conta:

- A unidade padrão de contagem relativa: **Ppq (Pulses per quarter note³⁷)**
- O parâmetro de conversão da unidade padrão de contagem relativa (Ppq) para tempo absoluto: o Meta Evento **Set Tempo**

A seguir serão apresentados estes dois fatores.

2.9.2.7 Ppq.

A **Ppq** é um valor definido por dois Bytes³⁸, o qual é registrado no cabeçalho principal de um arquivo MIDI SMF. A mesma determina qual será o valor padrão da contagem de uma semínima em um determinado arquivo MIDI (SMF).

Assim, se o SMF tiver uma Ppq de 120, isto significa, por exemplo, que o valor da contagem de tempo de uma semínima é igual a 120. Em consequência disto, tem-se que:

- Uma contagem de 240 equivaleria ao tempo de uma mínima (dobro da semínima);

³⁷ A Ppq, em alguns softwares, como o Cakewalk, é denominada também por Tic per quarter note.

³⁸ Com dois Bytes, tem-se a opção de contagens até 65.535. Quanto maior o valor de uma Ppq maior a definição do registro do tempo de cada evento MIDI. Assim, ao se capturar um evento, como o de notas musicais, pode-se registrar com precisão notas musicais de curtíssima duração, tornando o arquivo MIDI gravado mais fiel à interpretação original. Bons softwares de captura utilizam atualmente Ppqs de 1024, o que significa, uma definição para semifusas, de 64 contagens. Pode-se equivaler esta definição com a definição de uma imagem em *pixels*, quanto maior o número de pontos que definem uma imagem, melhor sua definição. Quanto maior a Ppq, maior a definição de uma música.

- Uma contagem de 60 equivaleria ao tempo de uma colcheia (metade da semínima).

Desta forma, com uma $Ppq = 120$, se um Delta Time de um evento possuir valor igual a 300, isto significa que o mesmo possui uma duração equivalente ao tempo de uma mínima (240) mais o tempo de uma colcheia (60), ou seja: $240 + 60 = 300$.

Portando, com este recurso, não se tem mais necessidade de conhecer a velocidade da máquina MIDI que gerou as contagens do arquivo SMF. Isto se dá devido ao fato de que o mesmo contador que conta o tempo de uma semínima (do padrão de conversão), contará, também, na mesma velocidade, o tempo de todos os eventos do SMF.

2.9.2.8 O cálculo do tempo de um evento em segundos.

De posse do tempo relativo de cada evento, necessita-se saber qual é o tempo, em segundos, de cada evento MIDI registrado nas mensagens dos SMF.

É interessante lembrar que em uma partitura CPN, o valor absoluto, em segundos, do tempo das notas musicais era obtido pelo conhecimento da figura musical desejada, associado ao valor do metrônomo adotado e registrado pelo músico na partitura.

Nos arquivos MIDI SMF, o tempo absoluto, em segundos, de um evento, é calculado determinando-se quantas semínimas equivale uma determinada contagem, e, de posse deste valor, deve-se multiplicar pelo tempo de cada semínima, conforme equação a seguir:

Tempo de uma contagem = (valor do Delta Time / Ppq) x (tempo de uma semínima)

O valor temporal, em segundos, da semínima, é registrado em um Meta Evento chamado: **Set Tempo**. Este Meta Evento informa o tempo da semínima em

microssegundos³⁹. Este tempo é registrado em 3(três) Bytes, levando em consideração os 8 (oito) bits de cada Byte.

2.9.2.9 Meta Evento Set Tempo.

O formato de um Meta Evento é como segue:

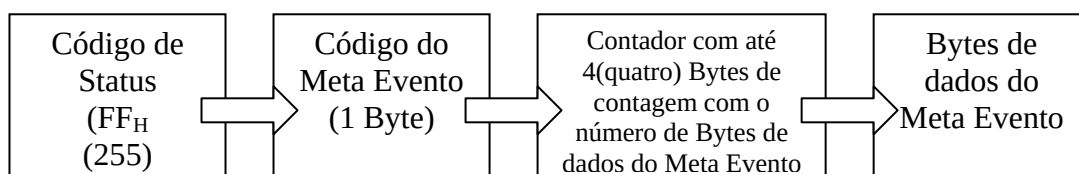


Figura 2.44 – Formato de um Meta Evento

- **Byte indicador de Meta Evento (status);**
- O **Código do Meta Evento de tempo**, Set Tempo, é 51_H (81₁₀);
- O número de Bytes do contador é 1(um);
- O número de **Byte de dados deste Meta Evento** é 3(três);
- O número de **Bytes do Meta Evento Set Tempo** é fixo e igual a 6(seis).

Assim, a mensagem do Meta Evento **SET TEMPO** para o tempo de um semínima igual a 1(um) segundo⁴⁰ é:

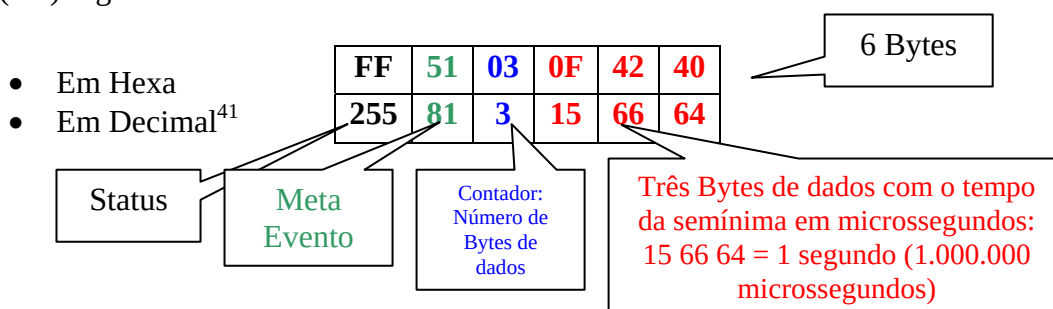


Figura 2.45 – Meta Evento Set Tempo

A forma com que este meta evento é armazenado no SMF será visto ainda neste capítulo no item 2.12

³⁹ Um microssegundo = 10^{-6} segundos = 0,000001 segundos

⁴⁰ Como o tempo é registrado em microssegundos, tem-se que 1.000.000 (10^6) microssegundos = 10^6 microssegundos = $10^6 \times 10^{-6}$ segundos = $10^{(6-6)}$ segundos = 10^0 segundos = 1segundo).

⁴¹ 15 66 64 = $15 \times 2^{16} + 66 \times 2^8 + 64 \times 2^0 = 15 \times 65536 + 66 \times 256 + 64 = 1.000.000$

2.9.2.10 Lendo uma mensagem com a máquina MIDI.

O processo de leitura de um SMF, independente do tipo de formato, segue os seguintes passos básicos:

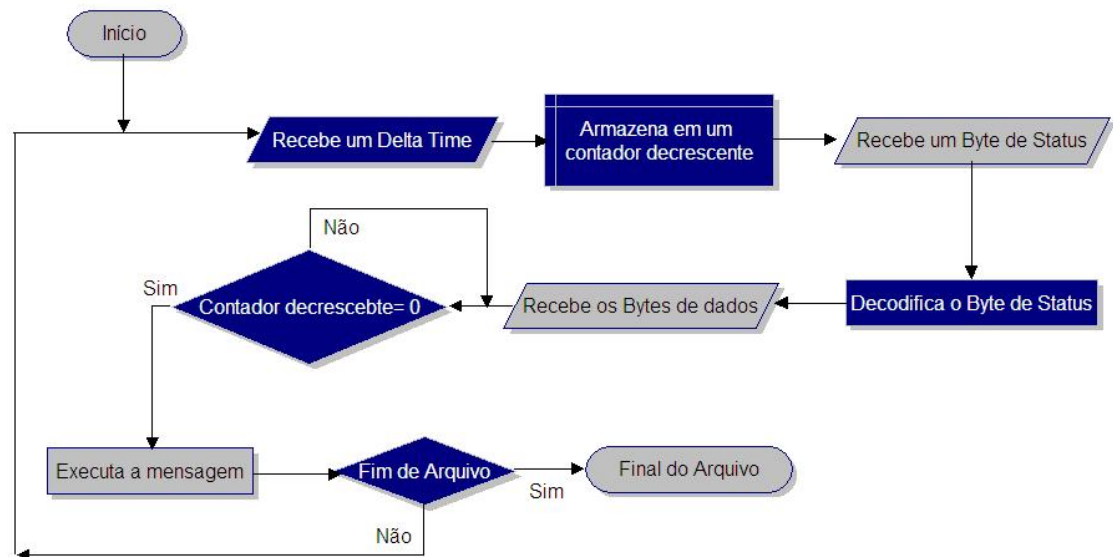


Figura 2.46 – Fluxograma – Lendo uma mensagem MIDI

Ou seja:

1. A máquina MIDI recebe um Delta Time (contagem de 1 a 4 Bytes)⁴² e a armazena em um contador decrescente (regressivo);
2. Recebe um Byte de status informando o tipo de evento que será executado, decodifica-o pra saber o formato do evento e quantos bytes a mensagem terá;
3. Após decodificado o evento e ajustado a máquina MIDI para executá-lo, lê todos os Bytes da mensagem;
4. Quando a contagem armazenada no contador regressivo chega a zero a máquina MIDI executa o evento da mensagem;
5. Recebe um novo Delta Time e repete os passos de 1 a 4, até que uma mensagem, o Meta Evento de fim de arquivo SMF (FF 2F 00), é recebida.

⁴² Este contador inicialmente, para o primeiro evento, geralmente possui o valor 0. Ou seja, não espere nenhum tempo. Recebe o primeiro Byte da primeira mensagem e o executa imediatamente.

2.9.2.11 O Contador regressivo.

O motivo de se ter um contador regressivo pode ser entendido, tomando como exemplo, novamente, a ação de ativar e desativar uma nota.

Quando um músico pressiona uma tecla de seu teclado musical, uma nota musical é ativada, começa a soar. Neste momento, internamente no teclado, um contador é disparado iniciando uma contagem relativa de tempo proporcional à duração deste evento. O contador continua tocando até que o tecladista retira o dedo da tecla, liberando-a. Ao fazer isto, a contagem é interrompida e armazenada e uma mensagem de desativar nota é enviada, cessando o som produzido inicialmente.

Assim, o ato de ativar e desativar nota registra os seguintes eventos:

1. Mensagem de ativar nota
2. Contagem do tempo de duração do evento
3. Mensagem de desativar nota.

Ao reproduzir esta mesma ação, a máquina MIDI lê a mensagem de ativar a nota e a ativa, logo após lê um Delta Time com a contagem do tempo de duração deste evento. Para que este evento, ativar nota, seja reproduzido com fidelidade, a máquina MIDI armazena esta contagem em um contador regressivo, com velocidade de contagem baseada na velocidade do processador, da Ppq e do Meta Evento Set Tempo. Assim, este contador vai decrementando a contagem com a mesma velocidade com que foi incrementada durante a geração do SMF. Quando a contagem chega a zero (tempo em que originalmente a nota musical ficou ativada), a máquina MIDI executa o próximo evento, ou seja, o de desativar nota, cessando o som da mesma.

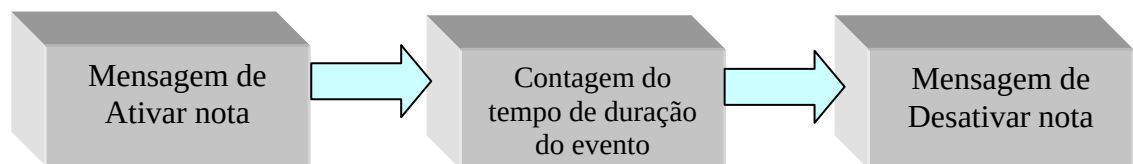


Figura 2.47 – Contador Regressivo

Observe, portanto, que, com estas ações: **Evento -> Delta Time -> Evento**, consegue-se reproduzir fielmente uma mensagem MIDI.

2.10 Diferença básica entre SMF formato 0 e formato 1.

O objetivo deste trabalho está focado na manipulação de arquivos e geração de aplicativos MIDI em SMF formato 0, mas, mesmo assim, é interessante que se tenha um conhecimento também do formato 1, já que estes dois formatos são os atualmente aceitos pela quase totalidade dos equipamentos MIDI (computadores, teclados). Um arquivo SMF armazena os eventos e Meta Eventos MIDI em forma de mensagens, tendo como base a estrutura da máquina MIDI, com suas mensagens e contagens relativas de tempo armazenadas em Delta Times.

A diferença básica entre os formatos 0 e 1 reside na forma com que os eventos disparados em cada canal MIDI são registrados em um arquivo SMF. Como já visto anteriormente, para se registrar uma sequência musical em SMF, deve-se:

- Armazenar a informação da unidade de tempo básica do arquivo: a Ppq;
- Armazenar os Meta Eventos necessários para que se possa reproduzir o arquivo no andamento fiel da música original, tal como o Meta evento de Set Tempo;
- Armazenar Meta Eventos de fórmula de compasso⁴³, armadura de clave⁴⁴(tonalidade), texto, copyright, lirismo, outros;
- Armazenar os eventos de voz, ou seja, os que atuam diretamente no som, nas notas musicais que serão produzidas por cada canal MIDI;
- Armazenar mensagens exclusivas de sistemas para determinado fabricante de um instrumento.

Assim, o que difere realmente o formato 0 e o formato 1 está, inicialmente, nos objetivos, no público alvo a quem se destina.

⁴³ Fórmula de compasso: Indica qual é a figura básica que será adotada como unidade de tempo da música e do metrônomo e, também, quantas unidades da mesma comporão um compasso musical: Exemplo: fórmula de compasso = 2 x 4 . Isto indica que a figura musical é a semínima (4) e que em cada compasso cabem 2 desta figura. Se fórmula de compasso fosse 3 x 2, isto indicaria que em cada compasso ter-se-ia o tempo de 3 mínimas(2). Onde: 1= semibreve, 2= mínima, 4 = semínima, 8 = colcheia, e assim por diante.

⁴⁴ Armadura de clave: indica os acidentes (notas com sustenido ou bemóis) da música, os quais definem a sua tonalidade. Como exemplo, na tonalidade dó maior não se tem acidentes, na tonalidade de sol maior só se tem o fá# como acidente [1],[2],[3].

2.10.1 Formato 0.

O formato 0 é destinado para teclados e equipamentos que vão ler o arquivo e executá-los praticamente em tempo real, ou seja, não precisa ler todo o arquivo e interpretá-lo para depois reproduzi-lo. Assim, no instante da geração dos arquivos SMF formato 0, os eventos são armazenados na ordem em que são gerados, independente de qual canal MIDI o tenha executado.

Observe a figura 2.48 a seguir:

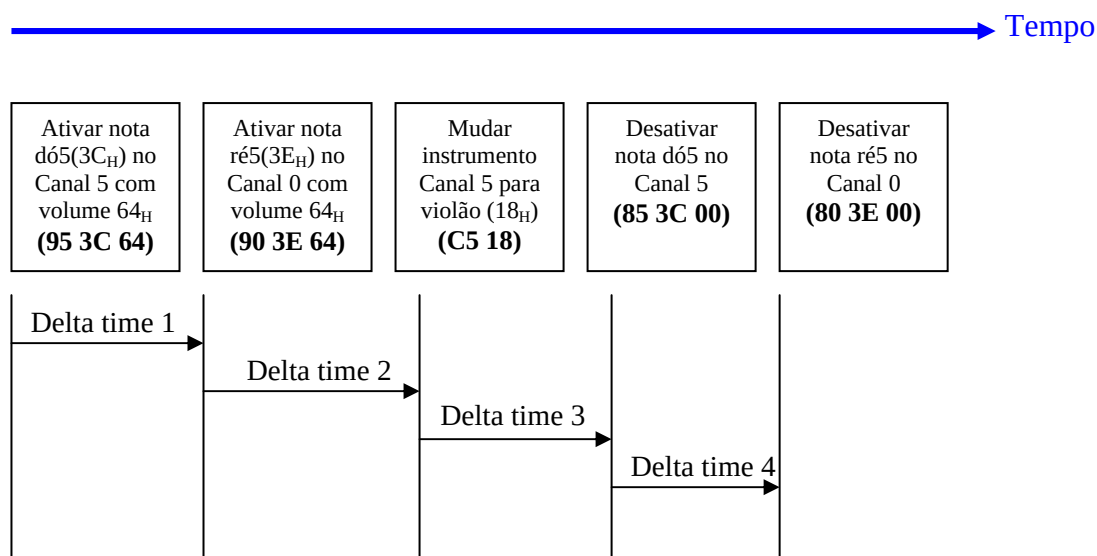


Figura 2.48 – Registro de mensagens no formato 0

No armazenamento das mensagens correspondentes aos eventos, as mesmas, independente do canal MIDI, seriam registradas em uma trilha (**Track**) de informação, na seqüência em que fossem ocorrendo, no caso:

Track musical único:

(95 3C 64) **DeltaTime1** (90 3E 64) **DeltaTime2** (C5 18) **DeltaTime3** (85 3C 00) **DeltaTime4** (80 3E 00)

Assim, quando a máquina MIDI for ler este arquivo, bastará ao mesmo executar cada evento na ordem em que aparecem, esperando a contagem de cada DeltaTime, precedente ao evento, chegar a zero antes de executá-los.

Portanto, nos SMF formato 0, todos os eventos de todos os canais são armazenados em seqüência em um único track de informação.

2.10.2 Formato 1.

O formato 1 é destinado principalmente para softwares de editoração de grades orquestrais⁴⁵ de partituras, onde, a princípio, o formato 0 é inapropriado, não aderente ao formato de uma partitura. Diz-se não aderente porque em uma grade orquestral tem-se registrado a partitura de todos os instrumentos musicais em pentagramas diferentes, conforme mostrado na Figura 2.49 a seguir:



Figura 2.49 – Grade Orquestral

Observe que em uma grade orquestral se tem o registro de todos os eventos separadamente, mas temporalmente dependentes, ou seja, sincronizados. No SMF formato 0 isto não ocorre, os eventos, registrados em mensagens MIDI, estão registrados em um só track de informação, sequenciadamente de acordo com que ocorrem, necessitando que sejam separados por instrumentos (canais MIDI), recalculando o tempo por canal para que se possa ter uma visão paralela, sincronizada, de todos os eventos ao mesmo tempo.

Assim, surge o formato 1, onde cada ocorrência de eventos de um canal MIDI (instrumento da grade orquestral), será registrado em uma ou mais trilhas (tracks) de informação independente. Diz-se uma ou mais trilhas devido ao fato de que em uma grade orquestral pode-se ter mais de um pentagrama de um mesmo instrumento, como

⁴⁵ Uma grade orquestral é uma partitura onde se tem, separadamente, em paralelo, todas as partituras dos instrumentos da música.

no caso de um quarteto de violões (4 violões tocando melodias diferentes, por exemplo). Desta forma, em um arquivo SMF formato 1, também pode-se ter mais de uma trilha de informação de **um mesmo instrumento** (canal), com conteúdos diferentes.



Figura 2.50 – Dois Instrumentos iguais em uma mesma grade

Isto significa que, mesmo que em uma máquina MIDI só se tenha 16 canais MIDI (instrumentos) diferentes, pode-se ter quantas trilhas (tracks) de gravação se desejar para cada um destes canais (instrumentos) para armazenar uma execução musical.

De posse das informações separadas de cada execução musical, um programa de editoração de partituras não terá dificuldade em *plotar* um pentagrama para cada uma das trilhas (tracks) musicais registradas. Observe como fica o registro em formato 1 do mesmo exemplo mostrado anteriormente para o registro de mensagens no formato 0.

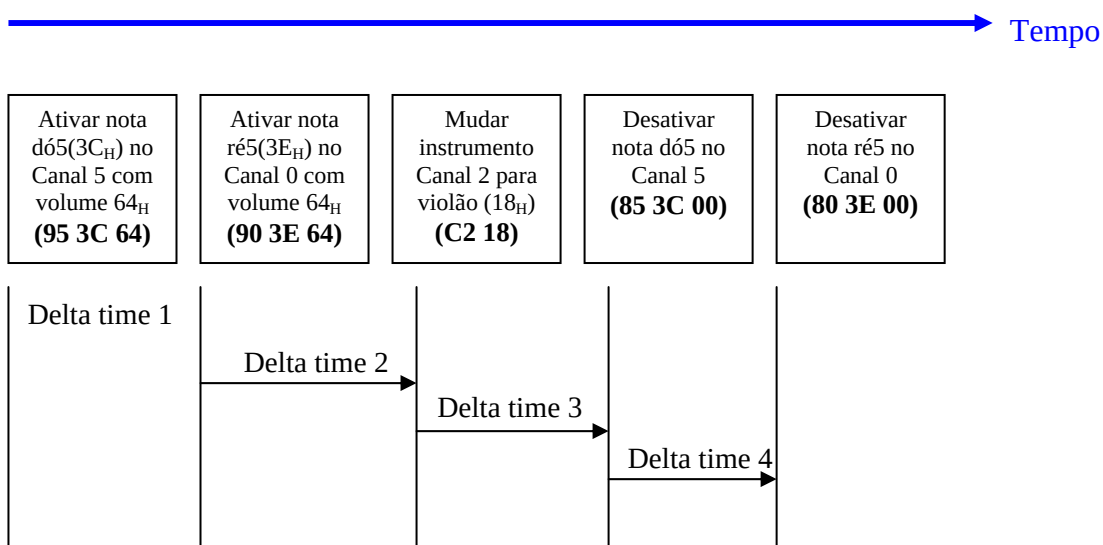


Figura 2.51 – Registro de mensagens no formato 1

Observe que ser tem eventos ocorrendo no canal MIDI 5, no canal MIDI 0 e no canal MIDI 2.

O registro destas informações em *tracks* diferentes ficaria:

Track1 (canal 0):

DeltaTime1 (90 3E 64) (**DeltaTime2** + **DeltaTime3** + **DeltaTime4**) (80 3E 00)

Track2 (canal 2):

(**DeltaTime1** + **DeltaTime2**) (C2 18)

Track3 (canal 5):

(95 3C 64) (**DeltaTime2** + **DeltaTime3**) (85 3C 00)

Assim, pode-se perceber que fica fácil extrair as informações temporais, por canal, de cada evento da música, a saber:

- Canal 0-> a melodia inicia com a nota musical ré5 (3E) após um tempo equivalente ao DeltaTime 1, durando a mesma o tempo equivalente à soma dos tempos dos DeltaTimes 2,3 e 4;
- Canal 2-> Nenhuma melodia, até então, foi iniciada, tendo somente ocorrido uma solicitação de mudança de instrumento para violão acústico;
- Canal 5-> a melodia inicia com a nota dó5 no início da leitura do arquivo, durando a mesma (nota dó5) equivalente ao tempo dos DeltaTimes 2 e 3.

2.11 O Arquivo MIDI SMF.

Visto como é a estrutura de armazenamento de mensagens em tracks, tanto no formato 0 quanto no formato 1, pode-se, agora, completar a estrutura final dos SMF, os quais, além de eventos musicais, possuem cabeçalhos e parâmetros iniciais que devem ser

registrados para que se possa reproduzir as músicas corretamente, com fidelidade às originais que lhes deram origem.

A seguir, é apresentada a estrutura dos arquivos SMF contendo as informações mínimas necessárias para o entendimento da hierarquia da mesma e a formatação dos SMF. Os códigos de status e dados são apresentados em hexadecimal para facilitar a visualização dos mesmos.

Os Arquivos SMF seguem a seguinte estrutura

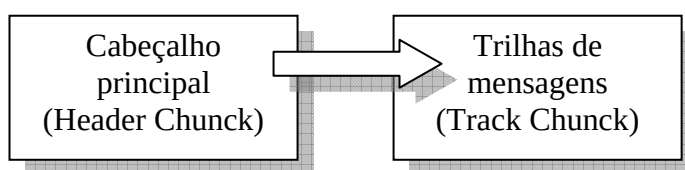


Figura 2.52 – Estrutura dos arquivos SMF

Tanto a estrutura do cabeçalho principal quanto a das trilhas de mensagens são iguais, o que diferencia os formatos é que no cabeçalho principal se tem a informação do tipo de formato e do número de tracks que o SMF possui, e que, nas trilhas de mensagens, no formato 1 só se devem ter mensagens de um mesmo canal.

2.11.1 Cabeçalho Principal (Header Chunk).

O cabeçalho principal, tanto para o formato 0 quanto para o formato 1, possuem um número fixo de Bytes igual a 14, separados da seguinte forma:

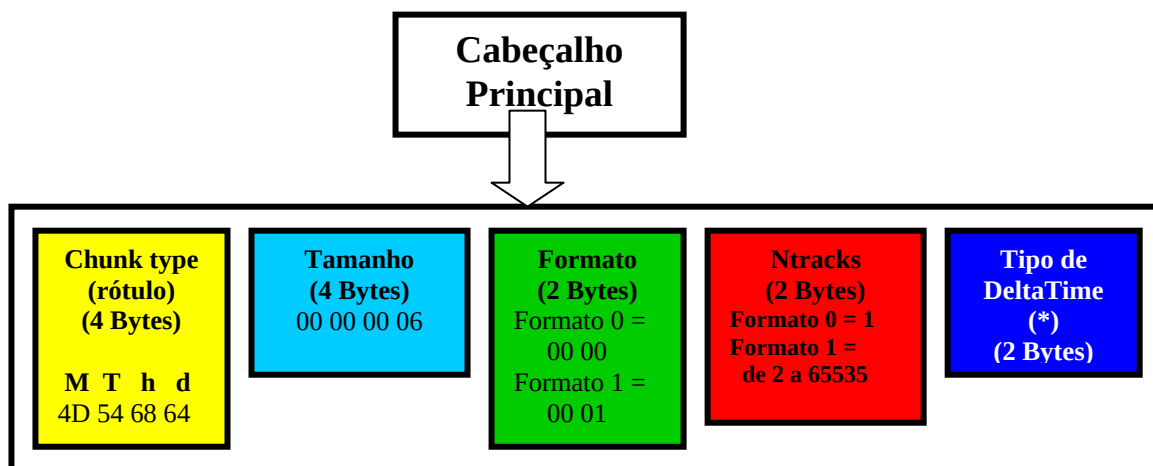


Figura 2.53 – Estrutura dos arquivos SMF -> Cabeçalho Principal

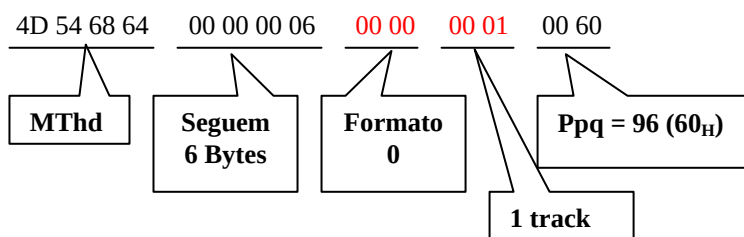
(*) – Se o bit mais significativo destes dois Bytes for igual a 0, isto indica que o tipo de DeltaTime é **Ppq** (pulsos por semínima), se for 1, a contagem dos DeltaTimes será feita no formato **SMPTE**⁴⁶ (frames por segundo)

Os dois exemplos a seguir ilustram com clareza a diferença do cabeçalho principal (header chunk) nos formato 0 e formato 1

Exemplo 1:

- Formato 0
- Ppq = 96

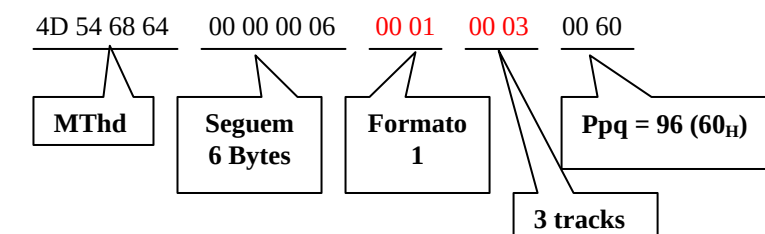
Tem-se



Exemplo 2:

- Formato 1
- 3 tracks
- Ppq = 96

Tem-se



Observe que no cabeçalho principal a única diferença existente entre o formato 0 e o formato 1 está na especificação do tipo de formato (Bytes 9 e 10) e na quantidade de tracks (Bytes 11 e 12).

⁴⁶ SMPTE- Society of **M**otion **P**icture and **T**elevision **E**ngineers – Padrão de contagem de tempo utilizada para sincronismo de som com imagem. A codificação de tempo **SMPTE** armazena a contagem em forma de **horas, minutos, segundos e frames** (formato HMSF utilizado em programas de seqüenciamento, como Cakewalk). Nesta dissertação não se utilizará este tipo de codificação de tempo, apenas a **Ppq**.

2.11.2 Trilhas de mensagens (Track Chunk).

A estrutura dos *Tracks Chunks* também é igual para o formato 0 e para o formato 1. A diferença, conforme já explicado anteriormente, é que no formato 0 só se tem um track de mensagens e no formato um, pode-se ter vários.

2.11.3 Estrutura da trilha de mensagem do Formato 0.

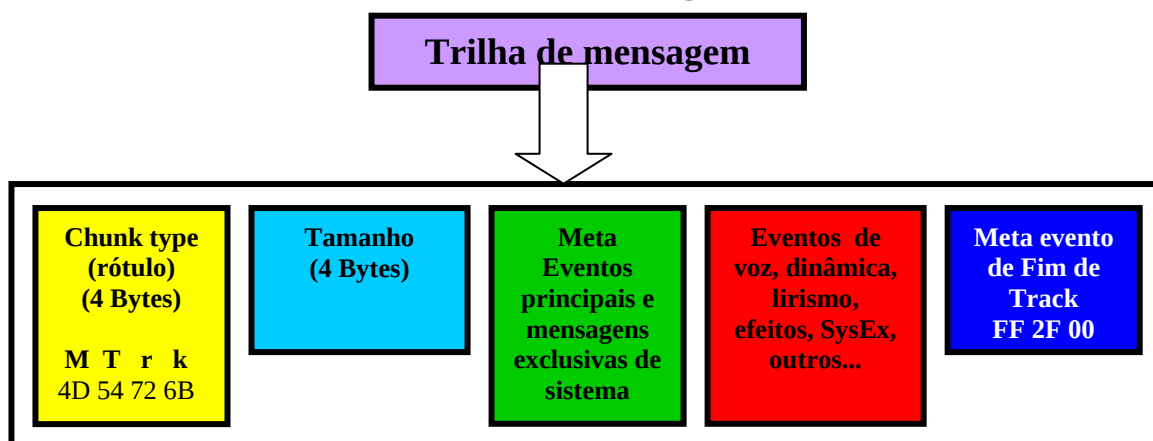


Figura 2.54 – Trilha única de um SMF formato 0

Para que um equipamento MIDI possa executar fielmente a seqüência musical gravada, o mesmo tem que ter informações adicionais além das mensagens de ativar e de desativar notas e das contagens (Delta Times) existentes. O mesmo necessita de pelo menos a informação do andamento, do tempo em segundos da unidade padrão do SMF (a semínima). É a partir desta mensagem que a máquina MIDI reproduz com fidelidade a música na mesma velocidade.

Como só se tem uma trilha, logo após o contador de Bytes do track⁴⁷ deve-se colocar os Meta Eventos e mensagens exclusivas de sistema principais do SMF, tais como:

- Andamento (Set Tempo);
- Armadura de Clave (tonalidade) – Key Signature;
- Fórmula de Compasso – Time Signature;
- Direitos Autorais (Copyright);
- Texto;
- Mensagens exclusivas (Sys Ex);
- Nome do Instrumento;
- Outros.

⁴⁷ O contador de bytes leva em conta todos os Bytes após o contador, incluindo os Bytes da mensagem do Meta Evento de Fim de Trilha.

Observações:

1. Quando não se registra o Meta Evento de tempo no SMF, as máquinas MIDI que forem ler o arquivo deve considerar o valor do metrônomo igual a 120, o que, em segundos dá 0,5 segundo por semínima;
2. Quando não se registra a fórmula de compasso e armadura de clave, as máquinas MIDI devem adotar a fórmula de compasso igual a 4x4 e a armadura de Clave da tonalidade dó maior;
3. Para se registrar uma pausa de uma determinada figura musical, basta colocar um Delta Time com contagem igual ao valor da figura desejada entre o ato de desativar uma nota e ativar outra (no exemplo de um arquivo formato 0 e 1, ainda neste tópico, esta característica será apresentada no item 2.10.1).

Logo após a inserção destas mensagens, as mensagens que registram a sequência musical propriamente dita são iniciadas. O *Track* termina quando a mensagem de Meta Evento de Fim de arquivo é encontrada (FF 2F 00).

2.11.4 Estrutura das trilhas de mensagens do Formato 1.

No formato 1, convencionou-se utilizar uma trilha, a primeira, para armazenar as mensagens exclusivas de sistema e as mensagens dos Meta Eventos principais de configuração e dados essenciais ao SMF.

Track 1 – Meta Eventos e Mensagens exclusivas de sistema

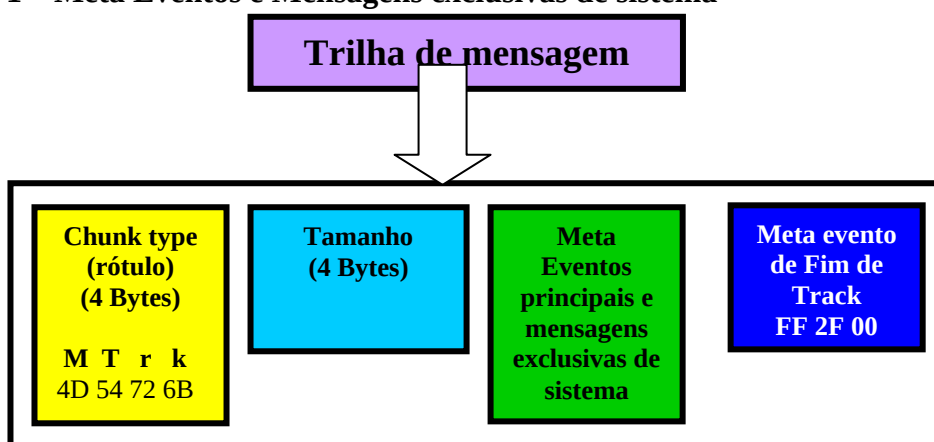


Figura 2.55 – Trilha de meta eventos e SysEx – formato 1

Demais Tracks – Cada track possui apenas informações de um canal MIDI

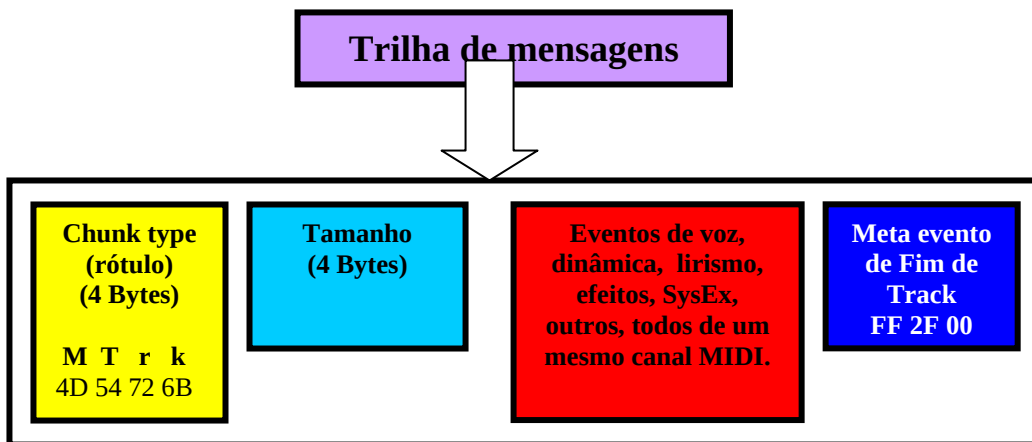


Figura 2.56 – Trilha de mensagem de canais – formato 1

2.12 Exemplo de uma seqüência musical a ser registrada em arquivo MIDI SMF Formato 0 e Formato 1.

Dados:

- **Número de instrumentos = 3**
 - Violão
 - Flauta 1
 - Flauta 2
- **Metrônomo = 120**
- **Ppq = 96**
- **Fórmula de compasso = 3x4**
- **Tonalidade = Sol maior (1 sustenido)**
- **Música:** segue a grade orquestral na *Figura 2.57* a seguir.

A grade orquestral mostra três partes musicais em 3/4 de Sol maior (uma chave de sustenido).
VIOLÃO: Começa com uma nota de Sol (G4) no primeiro tempo, seguida de uma pausa no segundo tempo e uma nota de Sol (G4) no terceiro tempo.
FLAUTA 2: Começa com uma pausa no primeiro tempo, seguida de uma escala descendente (F#4, E4, D4) no segundo tempo e uma pausa no terceiro tempo.
FLAUTA 1: Começa com uma nota de Sol (G4) no primeiro tempo, seguida de uma pausa no segundo tempo e uma nota de Sol (G4) no terceiro tempo.

Figura 2.57 – Grade orquestral exemplo para Formato 0 e Formato 1

2.12.1 Arquivo MIDI Formato 0 da figura 2.57, sem *running status*.

Arquivo SMF completo em Hexadecimal

```
4D 54 68 64 00 00 00 06 00 00 00 01 00 60 4D 54 72 6B 00 00 00 4F 00 FF 59 02 01
00 00 FF 51 03 07 A1 20 00 FF 58 04 03 02 18 08 00 C0 18 00 C1 49 00 91 3E 64 00
90 45 64 60 80 45 00 00 91 43 64 30 81 43 00 00 91 40 64 30 81 40 00 00 81 3E 00 00
90 45 64 00 90 48 64 60 80 45 00 00 80 48 00 00 FF 2F 00
```

Arquivo SMF completo em Decimal

```
77 84 104 100 0 0 0 6 0 0 0 1 0 96 77 84 114 107 0 0 0 79 0 255 89 2 1 0 0 255 81 3 7
161 32 0 255 88 4 3 2 24 8 0 192 24 0 193 73 0 145 62 100 0 144 69 100 96 128 69 0 0
145 67 100 48 129 67 0 0 145 64 100 48 129 64 0 0 129 62 0 0 144 69 100 0 144 72
100 96 128 69 0 0 128 72 0 0 255 47 0
```

Separando o Cabeçalho principal (Head Chunk) da trilha única (Track Chunk), tem-se:

Cabeçalho principal

Hexa -> 4D 54 68 64 00 00 00 06 00 00 00 01 00 60

Decimal -> 77 84 104 100 0 0 0 6 0 0 0 1 0 96

Onde:

Código em Decimal	Código em Hexa	Explicação
77 84 104 100	4D 54 68 64	Rótulo MThd : todo arquivo MIDI começa com este rótulo
0 0 0 6	00 00 00 06	Tamanho: número de Bytes que seguem = 6
0 0	00 00	Formato 0
0 1	00 01	1 track
0 96	00 60	Byte mais significativo <128 -> marcação do tempo em Ppq = 96 (60 _{Hexa})

Tabela 2.3 – Cabeçalho principal

Trilha única em Hexa

```

4D 54 72 6B 00 00 00 4F 00 FF 59 02 01 00 00 FF 51 03 07 A1 20 00 FF 58 04 03
02 18 08 00 C0 18 00 C1 49 00 91 3E 64 00 90 45 64 60 80 45 00 00 91 43 64
30 81 43 00 00 91 40 64 30 81 40 00 00 81 3E 00 00 90 45 64 00 90 48 64 60 80
45 00 00 80 48 00 00 FF 2F 00

```

Trilha única em Decimal

```

77 84 114 107 0 0 0 79 0 255 89 2 1 0 0 255 81 3 7 161 32 0 255 88 4 3 2 24 8 0 192 24
0 193 73 0 145 62 100 0 144 69 100 96 128 69 0 0 145 67 100 48 129 67 0 0 145 64
100 48 129 64 0 0 129 62 0 0 144 69 100 0 144 72 100 96 128 69 0 0 128 72 0 0 255 47
0

```

Onde:

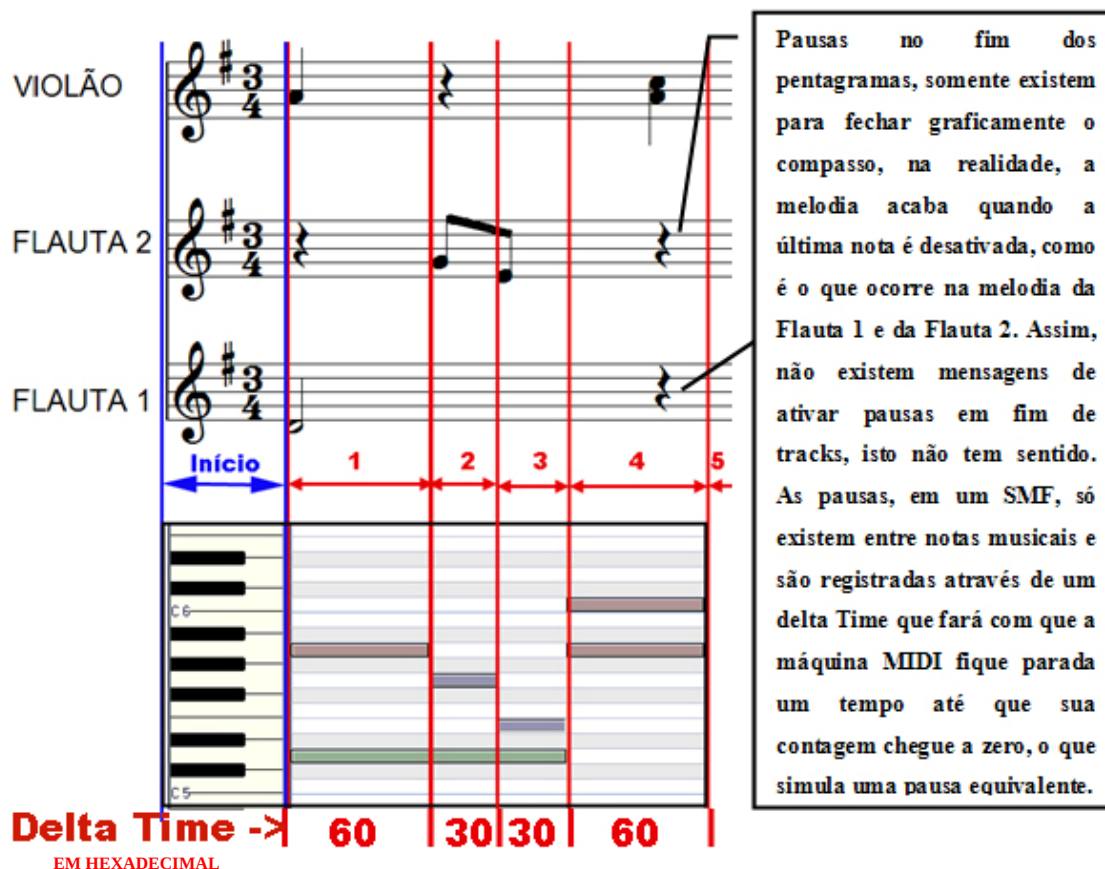


Figura 2.58 – Marcação para escrita da trilha do Formato 0

Os eventos e respectivas mensagens da seqüência musical devem ser registrados sequencialmente no formato 0 à medida que forem ocorrendo. Assim, a seqüência da trilha única do formato 0 fica:

Etapa	Ação	Código em Decimal	Código em Hexa	Explicação
Início	Colocar rótulo do Cabeçalho	77 84 114 107	4D 54 72 6B	Rótulo MTrk (Master Track): todo track MIDI começa com este rótulo.
Início	Colocar tamanho do track	0 0 0 79	00 00 00 4F	Tamanho = 79 (4F_H) Bytes. Número de Bytes que seguem, incluindo a mensagem de fim de track.
Início	Delta time 1	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
Início	Colocar Armadura de clave	255 89 2 1 0	FF 59 02 01 00	Meta Evento de Armadura de Clave FF = Status de Meta Evento 59 = Meta Evento Key Signature 02 = Número de Bytes de dados do Meta Evento 01 -> 1 sustenido 00 -> tonalidade maior -> tonalidade = Sol maior (Ver Anexo III)
Início	Delta time 2	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
Início	Colocar Set tempo	255 81 3 7 161 32	FF 51 03 07 A1 20	Meta Evento Set Tempo: FF = Status de Meta Evento 51 = Meta Evento Set Tempo 03 = Número de Bytes de dados do Meta Evento. O cálculo do tempo é dado nos três últimos Bytes do

				meta Evento. Dado o metrônomo, tem-se $\text{Tempo} = (60/\text{metrônomo}) \times 1000000 \text{ } \mu \text{ segundos} = 500000 \mu\text{s}$ Assim: $07 \text{ A1 } 20_{\text{H}} = 500000_{10} \text{ } \mu \text{ segundos}$
Início	Delta time 3	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
Início	Colocar Formula de compasso	255 88 4 3 2 24 8	FF 58 04 03 02 18 08	Meta Evento Fórmula de Compasso: Compasso 3x4 : FF = Status de Meta Evento 58 = Meta Evento Time Signature 04 = Número de Bytes de dados do Meta Evento 03 = 3 figuras por compasso 02 = figura musical = semínima = $4 (2^2)$ Os dois Bytes que seguem podem atualmente ter qualquer valor entre 0 e 127(7F), já que as máquinas MIDI atuais não necessitam mais destes dados. 18 = resolução: especifica o número de pulsos MIDI por tempo 08 = fusas por Compasso especifica o número de fusas em um compasso
Início	Delta time 4	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.

Início	Definir instrumento no Canal 0, violão	192 24	C0 18	Mensagem de Program Change (mudança de instrumento) C0 = mudar instrumento (C) no canal 0 (0) 18 = Violão
Início	Delta time 5	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
Início	Definir instrumento no Canal 1, flauta1	193 73	C1 49	Mensagem de Program Change (mudança de instrumento) C1 = mudar instrumento (C) no canal 1 (1) 49 = Flauta
1	Delta time 6	0	00	Delta time = 0 : início da música, o tempo ainda é zero ⁴⁸ .-> Executa a próxima mensagem imediatamente.
1	Ativar nota ré5 no canal 1, flauta 1	145 62 100	91 3E 64	Ativa nota ré5 (3E_H) no canal 1, 91 = ativa Nota no canal 1 3E = ré5 64 = Volume = 100 ₁₀
1	Delta time 7	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
1	Ativar nota lá5 no canal 0, violão	144 69 100	90 45 64	Ativa nota lá5 (45_H) no canal 0, 90 = ativa Nota no canal 0 45 = lá5 64 = Volume = 100 ₁₀
2	Delta time 8	96	60	Delta time = 96 (60_H) < 128(80 _H) -> 1 Byte->

⁴⁸ Tempo zero: Apesar dos Delta Times serem zero, na realidade os eventos iniciais não são executados todos ao mesmo tempo. O que ocorre é que os eventos são disparados um a um, em sequência, mas, como a máquina é rápida, tem-se a ilusão que o fato do Delta Time ser zero implica em que todos eles são executados ao mesmo tempo.

				Espera-se o tempo de uma semínima(60_H) para executar a próxima mensagem
2	Desativar nota lá5 do canal 0, violão	128 69 0	80 45 00	Desativa nota lá5 (45_H) do canal 0 80 = desativa Nota no canal 0 45 = lá5 00 = Volume
2	Delta time 9	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
2	Ativar nota sol5 do canal 1, flauta 2	145 67 100	91 43 64	Ativa nota sol5 (43_H) no canal 1, 91 = ativa Nota no canal 1 43 = sol5 64 = Volume = 100 ₁₀
3	Delta time 10	48	30	Delta time = Uma Colcheia (30_H): espera o tempo de uma colcheia para executar a próxima mensagem.
3	Desativar nota sol5 do canal 1, flauta 2	129 67 0	81 43 00	Desativa nota sol5 (43_H) do canal 1 81 = desativa nota no canal 1 43 = nota sol5 00 = volume
3	Delta time 11	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
3	Ativar nota mi5 no canal 1, flauta 2	145 64 100	91 40 64	Ativa nota mi5 (40_H) no canal 1, 91 = ativa nota no canal 1 40 = mi5 64 = volume = 100 ₁₀
4	Delta time 12	48	30	Delta time = Uma Colcheia (30_H): espera o tempo de uma colcheia para executar a

				próxima mensagem.
4	Desativar nota mi5 do canal 1, flauta 2	129 64 0	81 40 00	Desativa nota mi5 (40_H) do canal 1 81 = desativa nota no canal 1 40 = nota mi5 00 = volume
4	Delta time 13	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
4	Desativar nota ré5 do canal 1, flauta 1	129 62 0	81 3E 00	Desativa nota ré5 (3E_H) do canal 1 81 = desativa nota no canal 1 3E = nota ré5 00 = volume
4	Delta time 14	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
4	Ativar nota lá5 do canal 0, violão	144 69 100	90 45 64	Ativa nota lá5 (45_H) do canal 0, 90 = ativa nota no canal 0 45 = nota lá5 64 = volume=100 ₁₀
4	Delta time 15	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
4	Ativar nota dó6 do canal 0, violão	144 72 100	90 48 64	Ativa nota dó6 (48_H) do canal 0, 90 = ativa nota no canal 0 48 = nota dó6 64 = volume=100 ₁₀
5	Delta time 16	96	60	Delta time = 96 (60_H) < 128(80 _H) -> 1 Byte-> Espera-se o tempo de uma semínima(60_H) para executar a próxima mensagem
5	Desativar nota lá5 do	128 69 0	80 45 00	Desativa nota lá5 (45_H) do

	canal 0, violão			canal 0 80 = desativa nota no canal 0 45 = nota lá5 00 = volume
5	Delta time 17	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
5	Desativar nota dó6 do canal 0, violão	128 72 0	80 48 00	Desativa nota dó6 (48_H) do canal 0 80 = desativa nota no canal 0 48 = nota dó6 00 = volume
5	Delta time 18	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
5	Colocar Meta evento de fim de track	255 47 0	FF 2F 00	Meta Evento de Fim de track: Término da música

Tabela 2.4 – Sequência da trilha única do formato 0 sem running status

2.12.2 Arquivo MIDI Formato 0 da figura 2.58, com running status⁴⁹.

Arquivo SMF completo em Hexadecimal

```
4D 54 68 64 00 00 00 06 00 00 00 01 00 60 4D 54 72 6B 00 00 00 48 00 FF 59 02
01 00 00 FF 51 03 07 A1 20 00 FF 58 04 03 02 18 08 00 C0 18 00 C1 49 00 91
3E 64 00 90 45 64 60 80 45 00 00 91 43 64 30 43 00 00 40 64 30 40 00 00 3E 00
00 90 45 64 00 48 64 60 45 00 00 48 00 00 FF 2F 00
```

Arquivo SMF completo em Decimal

```
77 84 104 100 0 0 0 6 0 0 0 1 0 96 77 84 114 107 0 0 0 72 0 255 89 2 1 0 0 255 81 3 7
161 32 0 255 88 4 3 2 24 8 0 192 24 0 193 73 0 145 62 100 0 144 69 100 96 128 69 0 0
145 67 100 48 67 0 0 64 100 48 64 0 0 62 0 0 144 69 100 0 72 100 96 69 0 0 72 0 0 255
47 0
```

⁴⁹ Não se utiliza *running status* em Meta Eventos

Separando o Cabeçalho principal (Head Chunk) da trilha única (Track Chunk), tem-se:

Cabeçalho principal (não muda com *running status*)

Em Hexa -> 4D 54 68 64 00 00 00 06 00 00 00 01 00 60

Em Decimal -> 77 84 104 100 0 0 0 6 0 0 0 1 0 96

Trilha única (muda com *running status*)
Em Hexadecimal

4D 54 72 6B 00 00 00 47 00 FF 59 02 01 00 00 FF 51 03 07 A1 20 00 FF 58 04 03
02 18 08 00 C0 18 00 C1 49 00 91 3E 64 00 90 45 64 60 45 00 00 91 43 64 30 43
00 00 40 64 30 40 00 00 3E 00 00 90 45 64 00 48 64 60 45 00 00 48 00 00 FF 2F
00

Trilha única (muda com *running status*)
Em Decimal

77 84 114 107 0 0 0 71 0 255 89 2 1 0 0 255 81 3 7 161 32 0 255 88 4 3 2 24 8 0 192 24
0 193 73 0 145 62 100 0 144 69 100 96 69 0 0 145 67 100 48 67 0 0 64 100 48 64 0 0
62 0 0 144 69 100 0 72 100 96 69 0 0 72 0 0 255 47 0

Onde:

Etapa	Ação	Código em Decimal	Código em Hexa	Explicação apenas onde o <i>running status</i> ocorre e na contagem de Bytes do Track
Início	Colocar rótulo do Cabeçalho	77 84 114 107	4D 54 72 6B	
Início	Colocar tamanho do track	0 0 0 71	00 00 00 47	Tamanho = 71 (47 _H) (sem <i>running status</i> eram 79 Bytes)
Início	Delta time 1	0	00	
Início	Colocar Armadura de clave	255 89 2 1 0	FF 59 02 01 00	

Início	Delta time 2	0	00	
Início	Colocar Set tempo	255 81 3 7 161 32	FF 51 03 07 A1 20	
Início	Delta time 3	0	00	
Início	Colocar Formula de compasso	255 88 4 3 2 24 8	FF 58 04 03 02 18 08	
Início	Delta time 4	0	00	
Início	Definir instrumento no Canal 0, violão	192 24	C0 18	
Início	Delta time 5	0	00	
Início	Definir instrumento no Canal 1, flauta	193 73	C1 49	
1	Delta time 6	0	00	
1	Ativar nota re5 no canal 1, flauta 1	145 62 100	91 3E 64	
1	Delta time 7	0	00	
1	Ativar nota la5 no canal 0, violão	144 69 100	90 45 64	Status 90_H que servirá como running status a seguir
2	Delta time 8	96	60	
2	Desativar nota la5 do canal 0, violão	69 0	45 00	Utiliza-se o status corrente (Running Status) 90 . O status 80 é suprimido devido ao fato de que ativar uma nota com volume 0 é o mesmo que desativá-la. 80 45 00 = 90 45 00 = 45 00 (running status)
2	Delta time 9	0	00	
2	Ativar nota sol5 do canal 1, flauta 2	145 67 100	91 43 64	Status 91_H que servirá como running status a seguir
3	Delta time 10	48	30	

3	Desativar nota sol5 do canal 1, flauta 2	67 0	43 00	Utiliza-se o status corrente (Running Status) 91 _H . O status 81 _H é suprimido devido ao fato de que ativar uma nota com volume 0 é o mesmo que desativá-la. 81 43 00 = 91 43 00 = 43 00 (running status)
3	Delta time 11	0	00	
3	Ativar nota mi5 no canal 1, flauta 2	64 100	40 64	Utiliza-se o status corrente (Running Status) 91 _H .
4	Delta time 12	48	30	
4	Desativar nota mi5 do canal 1, flauta 2	64 0	40 00	Utiliza-se o status corrente (Running Status) 91 _H . O status 81 _H é suprimido devido ao fato de que ativar uma nota com volume 0 é o mesmo que desativá-la. 81 40 00 = 91 40 00 = 40 00 (running status)
4	Delta time 13	0	00	
4	Desativar nota re5 do canal 1, flauta 1	62 0	3E 00	Utiliza-se o status corrente (Running Status) 91 _H . O status 81 _H é suprimido devido ao fato de que ativar uma nota com volume 0 é o mesmo que desativá-la. 81 3E 00 = 91 3E 00 = 3E 00 (running status)
4	Delta time 14	0	00	
4	Ativar nota la5 do canal 0, violão	144 69 100	90 45 64	Status 90 _H que servirá como running status a seguir
4	Delta time 15	0	00	
4	Ativar nota do6 do	72 100	48 64	Utiliza-se o status corrente

	canal 0, violão			(Running Status) 90_H .
5	Delta time 16	96	60	
5	Desativar nota la5 do canal 0, violão	69 0	45 00	Utiliza-se o status corrente (Running Status) 90_H . O status 80_H é suprimido devido ao fato de que ativar uma nota com volume 0 é o mesmo que desativá-la. 80 45 00 = 90 45 00 = 45 00 (running status)
5	Delta time 17	0	00	Delta time = 0 -> Executa a próxima mensagem imediatamente.
5	Desativar nota do6 do canal 0, violão	72 0	48 00	Utiliza-se o status corrente (Running Status) 90_H . O status 80_H é suprimido devido ao fato de que ativar uma nota com volume 0 é o mesmo que desativá-la. 80 48 00 = 90 48 00 = 49 00 (running status)
5	Delta time 18	0	00	
5	Colocar Meta evento de fim de track	255 47 0	FF 2F 00	Meta Evento de Fim de track: Término da música

Tabela 2.5 – Sequência da trilha única do formato 0 com running status

2.12.3 Arquivo MIDI Formato 1 da mesma figura 2.58.

Arquivo SMF completo em Hexadecimal

```

4D 54 68 64 00 00 00 06 00 01 00 04 00 60 4D 54 72 6B 00 00 00 19 00 FF 59 02 01
00 00 FF 51 03 07 A1 20 00 FF 58 04 03 02 18 08 00 FF 2F 00 4D 54 72 6B 00 00 00
1F 00 C0 18 00 90 45 64 60 80 45 00 60 90 45 64 00 90 48 64 60 80 45 00 00 80 48 00
00 FF 2F 00 4D 54 72 6B 00 00 00 17 00 C1 49 60 91 43 64 30 81 43 00 00 91 40 64
30 81 40 00 00 FF 2F 00 4D 54 72 6B 00 00 00 10 00 C1 49 00 91 3E 64 81 40 81 3E
00 00 FF 2F 00

```

Arquivo SMF completo em Decimal

```

77 84 104 100 0 0 0 6 0 1 0 4 0 96 77 84 114 107 0 0 0 25 0 255 89 2 1 0 0 255 81 3 7
161 32 0 255 88 4 3 2 24 8 0 255 47 0 77 84 114 107 0 0 0 31 0 192 24 0 144 69 100 96
128 69 0 96 144 69 100 0 144 72 100 96 128 69 0 0 128 72 0 0 255 47 0 77 84 114 107
0 0 0 23 0 193 73 96 145 67 100 48 129 67 0 0 145 64 100 48 129 64 0 0 255 47 0 77
84 114 107 0 0 0 16 0 193 73 0 145 62 100 129 64 129 62 0 0 255 47 0

```

Separando o Cabeçalho principal (Head Chunk) das demais trilhas (Track Chunk), tem-se:

Cabeçalho principal em Hexadecimal

```
4D 54 68 64 00 00 00 06 00 01 00 04 00 60
```

Cabeçalho principal em Decimal

```
77 84 104 100 0 0 0 6 0 1 0 4 0 96
```

Onde:

Código em Decimal	Código em Hexa	Explicação
77 84 104 100	4D 54 68 64	Rótulo MThd : todo arquivo MIDI começa com este rótulo
0 0 0 6	00 00 00 06	Tamanho: número de Bytes que seguem = 6
0 1	00 01	Formato 1
0 4	00 04	4 tracks
0 96	00 60	Byte mais significativo <128 - > marcação do tempo em Ppq = 96 ₁₀ (60 _{Hexa})

Tabela 2.6 – Cabeçalho principal formato 1

Track 1 (Meta Eventos) em Hexadecimal

```
4D 54 72 6B 00 00 00 19 00 FF 59 02 01 00 00 FF 51 03 07 A1 20 00 FF 58 04 03 02
18 08 00 FF 2F 00
```

Track 1 (Meta Eventos) em Decimal

```
77 84 114 107 0 0 0 25 0 255 89 2 1 0 0 255 81 3 7 161 32 0 255 88 4 3 2 24 8 0 255 47
0
```

Onde:

O primeiro Track (**Track 1**) registra os Meta Eventos da sequência MIDI, assim a sequência do Track 1 fica:

Etapa	Ação	Código em Decimal	Código em Hexa	Explicação
Início	Colocar rótulo do Cabeçalho	77 84 114 107	4D 54 72 6B	Rótulo MTrk (Master Track): todo track MIDI começa com este rótulo.
Início	Colocar tamanho do track	0 0 0 25	00 00 00 19	Tamanho = 25 (19_H) Bytes. Número de Bytes que seguem, incluindo a mensagem de fim de track.
Início	Delta time 1	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
Início	Colocar Armadura de clave	255 89 2 1 0	FF 59 02 01 00	Meta Evento de Armadura de Clave FF = Status de Meta Evento 59 = Meta Evento Key Signature 02 = Número de Bytes de dados do Meta Evento 01 → 1 sustenido 00 → tonalidade maior → tonalidade = Sol maior (Ver Anexo III)

Início	Delta time 2	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
Início	Colocar Set tempo	255 81 3 7 161 32	FF 51 03 07 A1 20	Meta Evento Set Tempo: FF = Status de Meta Evento 51 = Meta Evento Set Tempo 03 = Número de Bytes de dados do Meta Evento O cálculo do tempo é dado nos três últimos Bytes do meta Evento. Dado o metrônomo, tem-se $\text{Tempo} = (60/\text{metrônomo}) \times 1000000 \text{ } \mu \text{ segundos} = 500000 \mu\text{s}$ Assim: $\text{07 A1 20}_H = 500000_{10} \text{ } \mu \text{ segundos}$
Início	Delta time 3	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
Início	Colocar Fórmula de Compasso	255 88 4 3 2 24 8	FF 58 04 03 02 18 08	Meta Evento Fórmula de Compasso: Compasso 3x4 ; FF = Status de Meta Evento 58 = Meta Evento Time Signature 04 = Número de Bytes de dados do Meta Evento 03 = 3 figuras por compasso 02 = figura musical = semínima = $4(2^2)$ Os dois Bytes que seguem podem atualmente ter qualquer valor entre 0 e 127(7F_H), já que as máquinas MIDI atuais não necessitam mais destes dados.

				18 = resolução: especifica o número de pulsos MIDI por tempo 08 = fusas por Compasso especifica o número de fusas em um compasso.
Início	Delta time 4	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
Início	Colocar Meta evento de fim de track	255 47 0	FF 2F 00	Meta Evento de Fim de track: Fim do Track 1

Tabela 2.7 – Eventos do track 1

Track 2 (Canal 0 = Violão) em Hexa

```
4D 54 72 6B 00 00 00 1F 00 C0 18 00 90 45 64 60 80 45 00 60 90 45 64 00 90 48 64 60
80 45 00 00 80 48 00 00 FF 2F 00
```

Track 2 (Canal 0 = Violão) em Decimal

```
77 84 114 107 0 0 0 31 0 192 24 0 144 69 100 96 128 69 0 96 144 69 100 0 144 72 100
96 128 69 0 0 128 72 0 0 255 47 0
```

Onde:

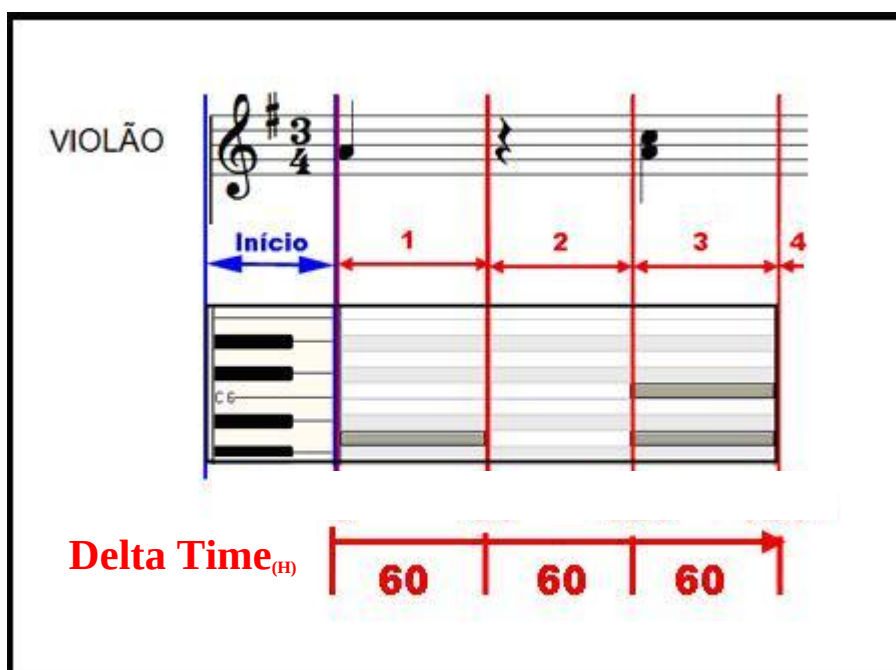


Figura 2.59 – Marcação para escrita da trilha do violão do Formato 1

O segundo Track (**Track 2**) registra as seqüências do Canal 0, o qual tem como instrumento o violão, ficando da seguinte forma:

Etapa	Ação	Código em Decimal	Código em Hexa	Explicação
Início	Colocar rótulo do Cabeçalho	77 84 114 107	4D 54 72 6B	Rótulo MTrk (Master Track): todo track MIDI começa com este rótulo.
Início	Colocar tamanho do track	0 0 0 31	00 00 00 1F	Tamanho = 31 (1F_H) Bytes. Número de Bytes que seguem, incluindo a mensagem de fim de track.
Início	Delta Time	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
Início	Definir instrumento no Canal 0, violão	192 24	C0 18	Mensagem de Program Change (mudança de instrumento) C0 = mudar instrumento (C) no canal 0 (0) 18 = Violão
1	Delta Time	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
1	Ativar nota lá5 no canal 0	144 69 100	90 45 64	Ativa nota lá5 (45_H) no canal 0, 90 = ativa Nota no canal 0 45 = lá5 64 = Volume = 100 ₁₀
2	Delta Time	96	60	Delta time = 96 (60_H) < 128(80_H) → 1 Byte → Espera-se o tempo de uma semínima(60_H) para executar a próxima mensagem
2	Desativar nota lá5 no canal 0	128 69 0	80 45 00	Desativa nota lá5 (45_H) do canal 0 80 = desativa Nota no canal 0 45 = lá5

				00 = Volume
3	Delta Time	96	60	Delta time = 96 (60_H) < 128(80_H) → 1 Byte → Espera-se o tempo de uma semínima(60_H) para executar a próxima mensagem
3	Ativar nota lá5 no canal 0	144 69 100	90 45 64	Ativa nota lá5 (45_H) no canal 0, 90 = ativa Nota no canal 0 45 = lá5 64 = Volume = 100 ₁₀
3	Delta Time	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
3	Ativar nota dó6 no canal 0	144 72 100	90 48 64	Ativa nota dó6 (48_H) do canal 0, 90 = ativa nota no canal 0 48 = nota dó6 64 = volume=100 ₁₀
4	Delta Time	96	60	Delta time = 96 (60_H) < 128 (80_H) → 1 Byte → Espera-se o tempo de uma semínima(60_H) para executar a próxima mensagem
4	Desativar nota lá5 no canal 0	128 69 0	80 45 00	Desativa nota lá5 (45_H) do canal 0 80 = desativa Nota no canal 0 45 = lá5 00 = Volume
4	Delta Time	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
4	Desativar nota dó6 no canal 0	128 72 0	80 48 00	Desativa nota dó6 (48_H) do canal 0 80 = desativa nota no canal 0 48 = nota dó6

				00 = volume
4	Delta Time	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
4	Colocar Meta evento de fim de track	255 47 0	FF 2F 00	Meta Evento de Fim de track: Fim do track 2

Tabela 2.8 – Eventos do track 2

Track 3 (Canal 1 = Flauta 2) em Hexa

4D 54 72 6B 00 00 00 17 00 C1 49 60 91 43 64 30 81 43 00 00 91 40 64 30 81 40 00 00 FF 2F 00

Track 3 (Canal 1 = Flauta 2) em Decimal

77 84 114 107 0 0 0 23 0 193 73 96 145 67 100 48 129 67 0 0 145 64 100 48 129 64 0 0 255 47 0

Onde:

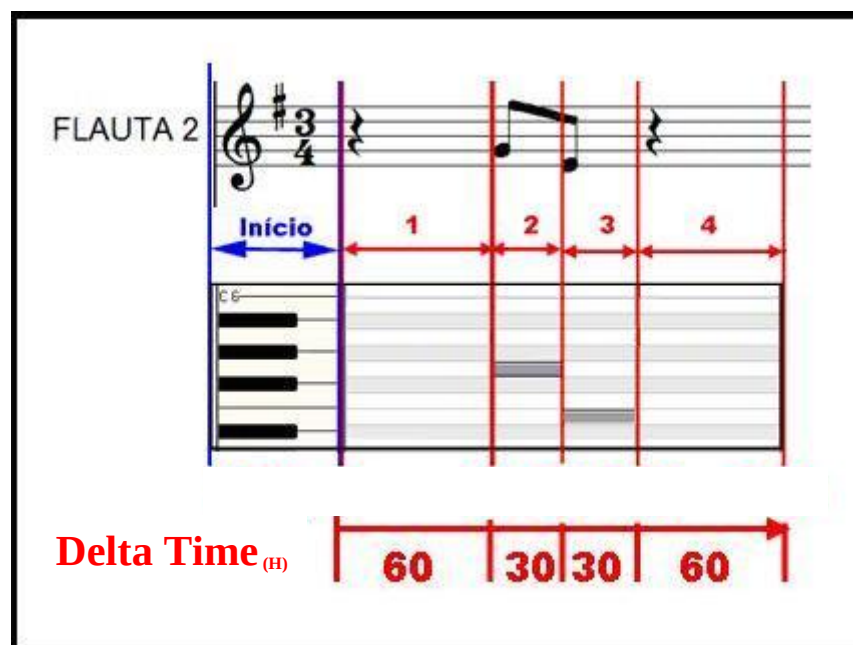


Figura 2.60 – Marcação para escrita da trilha da flauta 2 do Formato 1

O terceiro Track (**Track 3**) registra as seqüências do Canal 1, o qual tem como instrumento a flauta, ficando da seguinte forma:

Etapas	Ação	Código em Decimal	Código em Hexa	Evento
Início	Colocar rótulo do Cabeçalho	77 84 114 107	4D 54 72 6B	Rótulo MTrk (Master Track): todo track MIDI começa com este rótulo.

Início	Colocar tamanho do track	0 0 0 23	00 00 00 17	Tamanho = 23 (17_H) Bytes. Número de Bytes que seguem, incluindo a mensagem de fim de track.
Início	Delta Time	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
Início	Definir instrumento no Canal 1, Flauta	193 73	C1 49	Mensagem de Program Change (mudança de instrumento) C1 = mudar instrumento (C) no canal 1 (1) 49 = Flauta
1	Delta Time	96	60	Delta time = 96 (60_H) < 128 (80_H) → 1 Byte → Espera-se o tempo de uma semínima(60_H) para executar a próxima mensagem
2	Ativar nota sol5 do canal 1, flauta 2	145 67 100	91 43 64	Ativa nota sol5 (43_H) no canal 1, 91 = ativa Nota no canal 1 43 = sol5 64 = Volume = 100 ₁₀
3	Delta Time	48	30	Delta time = Uma Colcheia (30_H): espera o tempo de uma colcheia para executar a próxima mensagem.
3	Desativar nota sol5 do canal 1, flauta 2	129 67 0	81 43 00	Desativa nota sol5 (43_H) do canal 1 81 = desativa nota no canal 1 43 = nota sol5 00 = volume
3	Delta Time	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
3	Ativar nota mi5 do canal 1, flauta 2	145 64 100	91 40 64	Ativa nota mi5 (40_H) no canal 1,

				91 = ativa nota no canal 1 40 = mi5 64 = volume = 100 ₁₀
4	Delta Time	48	30	Delta time = Uma Colcheia (30_H): espera o tempo de uma colcheia para executar a próxima mensagem.
4	Desativar nota mi5 do canal 1, flauta 2	129 64 0	81 40 00	Desativa nota mi5 (40_H) do canal 1 81 = desativa nota no canal 1 40 = nota mi5 00 = volume
4	Delta Time	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
4	Colocar Meta evento de fim de track	255 47 0	FF 2F 00	Meta Evento de Fim de Track: Fim do track 3

Tabela 2.9 – Eventos do track 3

Track 4 (Canal 1 = Flauta 1) em Hexadecimal

```
4D 54 72 6B 00 00 00 10 00 C1 49 00 91 3E 64 81 40 81 3E 00 00 FF 2F 00
```

Track 4 (Canal 1 = Flauta 1) em Decimal

```
77 84 114 107 0 0 0 16 0 193 73 0 145 62 100 129 64 129 62 0 0 255 47 0
```

Onde:

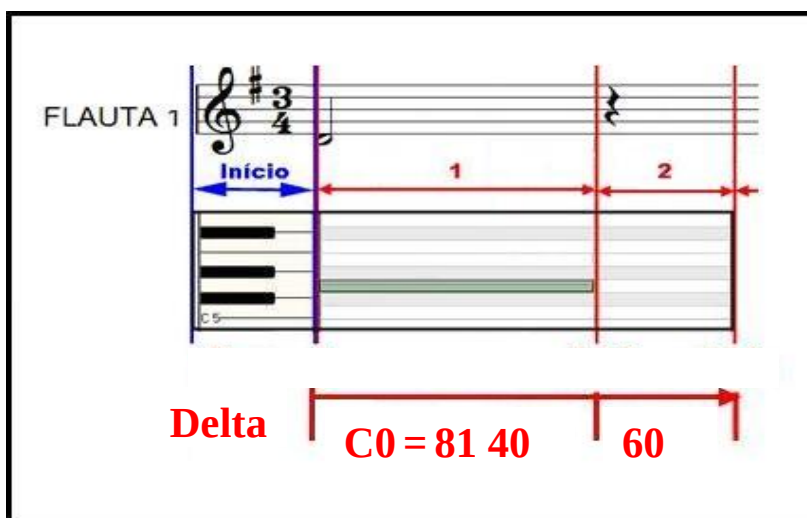


Figura 2.61 – Marcação para escrita da trilha da flauta 1 do Formato 1

O quarto Track (**Track 4**) registra as seqüências do Canal 1, o qual tem como instrumento a flauta, ficando da seguinte forma:

Etapa	Ação	Código em Decimal	Código em Hexa	Explicação
Início	Colocar Cabeçalho	77 84 114 107	4D 54 72 6B	Rótulo MTrk (Master Track): todo track MIDI começa com este rótulo.
Início	Colocar tamanho do track	0 0 0 16	00 00 00 10	Tamanho = 16 (10_H) Bytes. Número de Bytes que seguem, incluindo a mensagem de fim de track.
Início	Delta Time	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
Início	Definir instrumento no Canal 1, Flauta	193 73	C1 49	Mensagem de Program Change (mudança de instrumento) C1 = mudar instrumento (C) no canal 1 (1) 49 = Flauta
1	Delta Time	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
1	Ativar nota ré5 do canal 1, flauta 1	145 62 100	91 3E 64	Ativa nota ré5 (3E _H) no canal 1, 91 = ativa Nota no canal 1 3E = ré5 64 = Volume = 100 ₁₀
2	Delta Time	129 64	81 40	Delta Time = Uma mínima Contagem da mínima = 192 127 (1 Byte) < 192 < 16383 (Dois Bytes) → Dois Bytes de Delta Time = 10000001 01000000 = (81 40) _{Hexa} Espera-se o tempo de uma mínima (81 40_H) para executar a próxima mensagem

2	Desativar nota ré5 do canal 1, flauta 1	129 62 0	81 3E 00	Desativa nota ré5 (3E_H) do canal 1 81 = desativa nota no canal 1 3E = nota ré5 00 = volume
2	Delta Time	0	00	Delta time = 0 → Executa a próxima mensagem imediatamente.
2	Colocar Meta evento de fim de track	255 47 0	FF 2F 00	Meta Evento de Fim de Track: Fim do track 4

Tabela 2.10 – Eventos do track 4

2.13 Conclusão.

Este capítulo foi apresentado a um grupo de 3 alunos interessados em aprender sobre o protocolo MIDI e leitura e escrita de SFM formato 0 e formato 1, e, no final do estudo do mesmo, todos os alunos conseguiram ler os arquivos a eles destinados. Os mesmos também conseguiram escrever alguns arquivos SMF (formato 0 e 1) utilizando um dos editores implementados nesta dissertação (apresentado no capítulo de Estudos de casos, item 4: “Conversor Editor Didático MIDI Decimal/Hexadecimal”). Um destes arquivos utilizados pelo grupo teste foi o mostrado como exemplo neste capítulo. A seguir, são mostradas duas figuras: A primeira mostra a grade orquestral em formato zero, editada no Conversor Editor Didático, juntamente com o arquivo salvo no mesmo e lido por um software profissional, validando o aprendizado do grupo e a fidelidade do editor.

1- Grade orquestral editada e salva em formato 0.

Observe que a grade possui 3 tracks, mas dois são do mesmo canal, e, assim, são abertas como dois tracks, um por canal. É apresentada, também, a partitura em forma textual, um recurso do editor implementado.

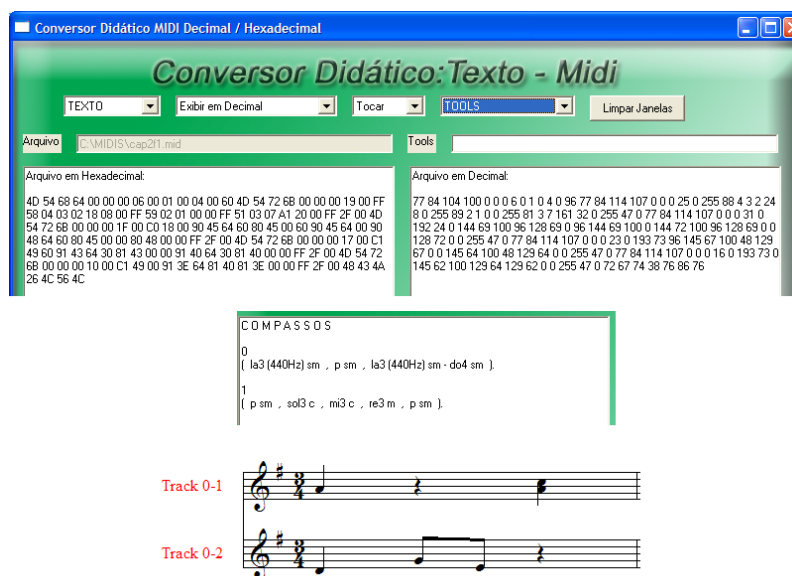


Figura 2.62 – Grade orquestral salva em formato 0

2- 2- Grade orquestral editada e salva em formato 1.

Mesmo possuindo a grade apenas dois canais, no formato 1 o editor profissional, no caso o Finale, abre cada track em um pentagrama, conforme configuração feita no mesmo.

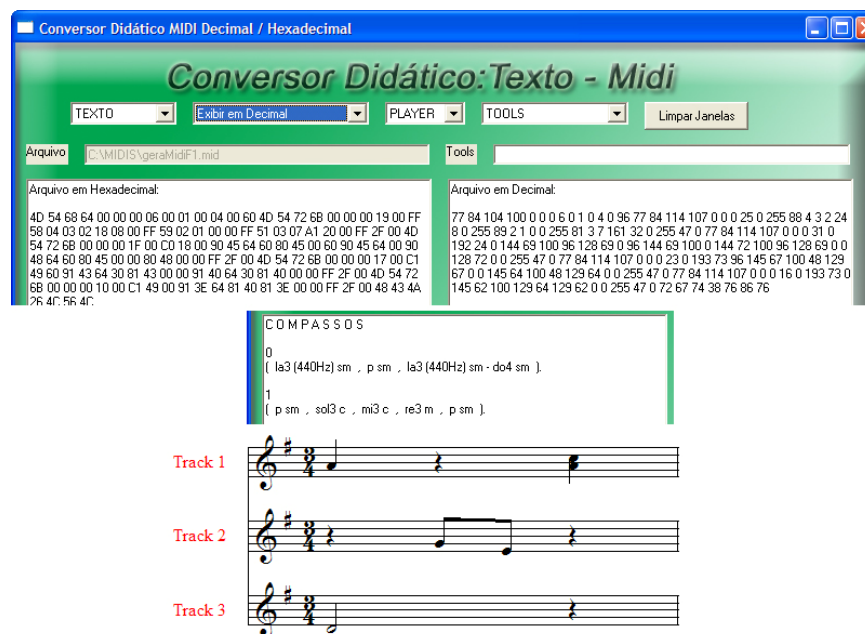


Figura 2.63 – Grade orquestral salva em formato 1

Conclui-se, portanto, que os objetivos deste capítulo - Apresentar didaticamente o protocolo MIDI e estrutura dos SMF, foram atingidos conforme desejado.

Capítulo 3

Linguagem Funcional Clean e Implementações Básicas.

Este capítulo é responsável por apresentar o paradigma e linguagem de programação escolhidos para o desenvolvimento de aplicativos em computação musical. Definida a linguagem, é também apresentado um pequeno resumo dos recursos básicos a serem utilizados com a mesma, bem como suas potencialidades. Isto é feito devido ser um dos objetivos, capacitar o leitor a uma base para desenvolver novos aplicativos MIDI utilizando a linguagem escolhida: CLEAN.

3.1 A Escolha pela Linguagem Clean.

A escolha de uma linguagem de programação adequada ao desenvolvimento de aplicativos MIDI foi um dos pontos fundamentais para se iniciar este trabalho. Inicialmente três pontos básicos deveriam ser cobertos na escolha, a saber:

- A linguagem escolhida deveria ser a mais aderente [2] possível ao profissional da música e de computação ao mesmo tempo;
- O aplicativo gerado pela linguagem deveria ser fácil de instalar, sem depender da instalação conjunta de dlls e *frameworks* que exigissem constantes atualizações do código;
- Ser livre.

Dos vários paradigmas de programação existentes, a escolha recaiu no paradigma lógico ou no funcional. Analisadas algumas linguagens destes paradigmas e os objetos do domínio musical que seriam modelados, definiu-se pelo paradigma funcional como base

de desenvolvimento dos aplicativos, principalmente pela facilidade de se implementar problemas recursivos em listas e vetores utilizando funções e funtores relativamente complexos. Neste paradigma, a linguagem Clean [2],[6],[7], foi a escolhida por possuir as seguintes características:

1. Por ser de código aberto e gratuito (pode-se baixá-la em: <http://clean.cs.ru.nl/>);
2. Por possuir alto nível de abstração na implementação de funções, evitando algoritmos complexos para modelagem das funções;
3. Por possuir funções matemáticas de alta ordem, como **map** e **fold**, que permitem aplicar uma função a um domínio completo e complexo de dados;
4. Por aceitar funções como parâmetro de outras funções, fatos comuns nas abstrações em música;
5. Por possuir implementação de notação Zermelo-Fraenkel [2],[5]. (http://en.wikipedia.org/wiki/Zermelo%E2%80%93Fraenkel_set_theory), a qual permite se definir o conjunto imagem diretamente da especificação da função e de seu domínio;
6. Por possuir transparência referencial, ou seja, o resultado de uma função, com mesmos argumentos, sempre dará o mesmo resultado, independentemente do contexto;
7. Por minimizar efeitos colaterais, mesmo em interfaces gráficas (utilizando a técnica de Tipos Únicos). Nesta técnica, tipos únicos, para se evitar a perda da transparência referencial, quando uma variável ou um arquivo ou outro registro do sistema estiver sendo utilizado, o mesmo é indisponibilizado a qualquer usuário ou sistema, até que uma nova instância do mesmo seja disponibilizada e liberada por quem iniciou sua manipulação. Assim, pode-se até fazer avaliações destrutivas nesta etapa, já que ela não será vista por mais ninguém além do processo que a está manipulando;
8. Por ser fortemente tipada, evitando que programas com problemas de tipo sejam compilados e, desta forma, não transferindo tal erro para avaliação pelo usuário. Assim, da mesma forma com que a matemática procede, uma função apenas manipula argumentos de mesmo tipo de dado. Estruturas computacionais, como listas e vetores, também só poderão ter elementos de mesmo tipo, podendo-se, desta forma, utilizá-la como domínio e conjuntos de dados a serem manipulados com segurança pelas funções desejadas;

9. Por não aceitar avaliação destrutiva de variáveis⁵⁰. Desta forma, evita-se que o programador tenha que monitorar todos os processos que utilizem a mesma variável, permitindo que se possa modularizar projetos com vários programadores e reaproveitar código de outros programas. Assim, uma vez que a variável é instanciada, recebe o valor, o mesmo se mantém durante todo o processamento do aplicativo;
10. Por possuir avaliação lazy⁵¹, e, quando desejado, avaliação eager⁵². Com a avaliação lazy, o programa apenas avalia uma expressão ou variável quando for necessário. Assim, por exemplo, pode-se conhecer o primeiro elemento de uma lista ou de um vetor sem que se tenha de conhecê-lo como um todo, agilizando a execução do programa. Caso se deseje utilizar avaliação eager, como no caso das linguagens procedurais, o Clean também disponibiliza tal recurso. Em alguns casos é até desejado que isto seja possível, como, no caso, de se querer validar toda uma lista de dados antes de se iniciar seu processamento, sem que seja necessário fazer uma função para tal procedimento;
11. Por possuir coleta automática de lixo, o que, mesmo para programadores experientes, é uma tarefa difícil e trabalhosa de ser implementada com eficiência. O ato de utilizar e liberar memória durante o processamento de um aplicativo é uma tarefa que demanda um bom conhecimento de software e do hardware (da máquina) onde o sistema irá rodar;
12. Por ser polimórfica⁵³ (Anexo VII), permitindo uma mesma função, em diferentes contextos, seja aplicada de forma diferente aos dados recebidos. Esta característica, em música é relevante. Música, como toda arte, é fortemente dependente de contexto;
13. Por permitir a utilização de códigos gerados por outras linguagens (obj, dll), como, por exemplo: C [6]. Assim, evita-se ter que desenvolver programas já

⁵⁰ A avaliação destrutiva só é aceita em interfaces gráficas, onde é necessária, mas tutelada pela técnica de tipos únicos que evita que tal avaliação cause efeitos colaterais e a perda da transparência referencial.

⁵¹ Lazy – do inglês: preguiçosa. Um termo que, traduzido, não expressa a intenção. Laze, no caso, significa que a linguagem não avalia os dados ou funções enquanto não for necessário. Isto não é preguiça, é eficiência. (http://en.wikipedia.org/wiki/Lazy_evaluation)

⁵² Eager – do inglês – ávida, desejada. No caso, o termo seria melhor a tradução precoce, prévia. A linguagem avalia todos os dados antes de manipulá-los, mesmo que a maioria deles não venha a ser utilizada. (http://en.wikipedia.org/wiki/Eager_evaluation)

⁵³ Não confundir polimorfismo com sobrecarga de operadores. Na sobrecarga apenas se define como um operador ou função vai manipular um determinado tipo de dado. A sobrecarga não se preocupa com o contexto. Uma função polimórfica, por trabalhar com dados de tipos diferentes, necessita que os operadores utilizados por ela sejam sobrecarregados para os tipos utilizados.

eficientes e consagrados por outras linguagens, principalmente quando se tem que desenvolver *drivers* para controle de placas e programas para manipular portas de dados, os quais demandam programação em baixo nível de abstração, como no caso de enviar e ler dados da placa de som;

14. Por ser, dentre todas as linguagens e paradigmas, uma das quatro linguagens mais eficientes e rápidas em quase todos os tipos de aplicações, usando vários *benchmarks* conhecidos e consagrados. No paradigma funcional escolhido, o Clean é a que possui melhor *benchmark*, superando *Eiffel*, *Haskell* e o mesmo Lisp (Anexo VIII)

(<http://shootout.alioth.debian.org/gp4/benchmark.php?test=all&lang=all>);

15. Por possuir código legível e limpo, mesmo em interfaces gráficas e visuais;

Além destas quinze características apresentadas, outra, não menos atraente, a 16^a, é a que se pôde constatar durante os estudos realizados: uma baixa razão de custo/aprendizado. Para uma pessoa que nunca programou, como foi o caso do autor desta dissertação, programar nesta linguagem é simples e aderente aos conceitos matemáticos adquiridos no ensino médio e aderente ao modelamento dos objetos e conceitos do domínio musical. Com tais conceitos e um bom raciocínio lógico pode-se programar em Clean a solução da maioria dos problemas encontrados no domínio musical. Neste domínio, geralmente, se tem um conjunto de dados de mesmo tipo e se deseja aplicar uma função de transformação ou inferência nos mesmos, tendo, como resposta, um novo conjunto de dados. Esta é praticamente a definição que os filósofos da matemática deram, a este processo: **Computar o valor de uma função**⁵⁴ [2]. A seguir, serão apresentados alguns conceitos e ferramentas utilizados nesta linguagem, de forma a permitir que interessados no desenvolvimento de aplicativos utilizando esta linguagem possam ter uma visão geral da mesma e de suas potencialidades.

3.2 Aderência e Notação Zermelo-Fraenkel: Notas da escala de dó maior.

Como primeiro exemplo de aderência, justificando a escolha desta linguagem para trabalhar com música, imagine ter um conjunto de notas musicais: **{dó,dó#,réb,ré,ré#,mib,mi,fá,fá#,sol,sol#,lá,lá#,si}**, e que deseja extrair do mesmo

⁵⁴Alonzo Church, em 1941, foi quem propôs o primeiro sistema formal de computação.

apenas as notas da escala de dó maior: **{dó,ré,mi,fá,sol,lá,si}**. Matematicamente, tem-se como domínio o conjunto de todas as notas musicais de uma determinada oitava musical⁵⁵. Para se obter o conjunto solução do problema, o conjunto imagem: as notas da escala de dó maior deve-se aplicar uma função que pegue como argumento cada elemento do domínio e o coloque no conjunto imagem (solução). Para que este elemento, a nota musical, seja aceito, o mesmo deve satisfazer à restrição de que ele deve pertencer à escala de dó maior. Assim, para pertencer ao conjunto solução (imagem), a nota musical não poderá ter acidentes, ou seja: não pode terminar com o caractere (símbolo) ‘b’ ou ‘#’, ou seja:

$$\left\{ \begin{array}{l} \textbf{Função} : f(x) = x \\ \textbf{Domínio} : x \in \text{domínio das notas musicais (Nm), onde:} \\ \quad \text{Nm} = \{\text{dó,dó\#,réb,ré,ré\#,mib,mi,fá,fá\#,sol,sol\#,lá,lá\#,si}\} \\ \textbf{Restrição do domínio} : \text{não possuir acidentes (terminar com o símbolo b ou \#)} \end{array} \right.$$

A formalização do conjunto imagem é dada pela notação Zermelo-Fraenkel, como segue:

$$\textbf{Conjunto solução} = \textbf{Imagem} = \{ \underset{\substack{\uparrow \\ \textbf{Função} \\ \text{ou regra}}}{f(x)} \mid \underset{\substack{\uparrow \\ \textbf{Domínio}}}{x \in \text{Nm}} \wedge \underset{\substack{\uparrow \\ \textbf{Restrição do Domínio}}}{x \text{ não possui acidentes}} \}$$

Implementar esta solução, em Clean, neste caso é totalmente aderente à especificação matemática mostrada, ou seja:

Matemática	->	{	f(x)		x ∈ Nm	∧	x não possui acidentes	}
			↓		↓		↓	
Clean	->	{	x	\	x <-: Nm		semAcidentes x	}

Observe que as definições são idênticas, apenas com uma sintaxe com símbolos diferentes, onde:

$$\boxed{| = \|} \quad , \quad \boxed{\in = <-:} \quad \text{e} \quad \boxed{\wedge = |}$$

⁵⁵ Oitava: um conjunto formado por doze notas musicais. Na quinta oitava, ter-se-iam as notas musicais: **dó5, dó#5, réb5, ré5, ré#5, mib5, mi5, fá5, fá#5, sol5, sol#5, lá5, lá#5, si5**.

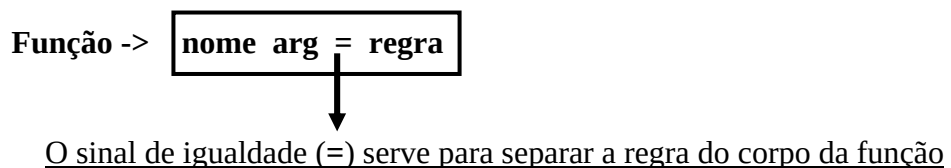
O único ponto que pode ser inicialmente encarado como não aderente, seria ter que implementar no Clean a função “**semAcidentes x**”, em $\{x \mid x <:- Nm \mid \text{semAcidentes } x\}$ a qual define o que é uma nota sem acidentes.

Por outro lado, caso se fosse solicitar que alguém também interpretasse a restrição “**x não possui acidentes**” em $\{f(x) \mid x \in Nm \wedge x \text{ não possui acidentes}\}$, na formalização matemática, caberia ao intérprete também ser informado do significado do que é uma nota musical não possuir acidentes. Assim, os dois modelos são similares e aderentes.

Implementar uma função que defina o que é uma nota musical sem acidentes, em Clean, também é uma tarefa tão aderente quanto o ato de se explicar tal conceito a uma pessoa que não conhece música. Para tanto, dever-se-á explicar à pessoa e ao sistema computacional que, para uma nota musical pertencer à escala de dó maior, a mesma não poderá possuir como último caractere, o caractere ‘b’ ou ‘#’.

3.2.1 Implementando uma função em Clean que testa se o último caractere de uma palavra, de uma *string*, é ou não um acidente musical.

Para se criar uma função deve-se recordar que a mesma deve possuir um nome, um ou mais argumentos e uma ou mais regras.



Exemplo de função com apenas uma regra: a função **dobro**

dobro x = 2*x

onde:

- **dobro** é o nome da função
- **x** é o argumento da função
- **2*x** é a regra da função

Exemplo de função com duas regras: a função **fatorial**

fatorial n

| n == 0 = 1

| n > 0 = n*fatorial (n-1)

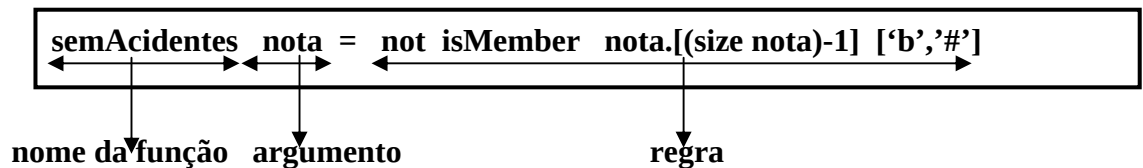
onde:

- **Fatorial** é o nome da função
- **n** é o argumento da função
- **1** é a primeira regra da função, denominada regra de parada
((**n = 0**) é a condição para que a regra **1** seja aplicada, a barra vertical (uma guarda) | indica que após ela existe uma condição ou restrição a ser obedecida).
- **n*fatorial (n-1)** é a segunda regra da função, denominada regra de recursão (**(n > 0)** é a condição para que a segunda regra seja aplicada).

A função **semAcidentes** é uma função com apenas uma regra e pode ser implementada, em Clean, da seguinte forma:

- **semAcidentes nota = not isMember nota.[(size nota)-1] ['b','#']**

Onde:



Ou seja:

- **nota** não possui acidentes (**semAcidentes nota**) se o último elemento do mesmo (**nota.[(size nota) - 1]**) não for membro (**not isMember**) da lista contendo os caracteres b e # (**['b','#']**)⁵⁶.

Colocando a regra da restrição do domínio na notação Zermelo-Fraenkel, a solução do problema proposto se mostra simples e legível, tanto para um músico quanto a um

⁵⁶ - **not** é a função de negação primitiva do Clean

- **isMember** é a função primitiva do Clean que testa se um elemento é membro de uma lista
o operador **.[]** pega um elemento de um vetor(uma string)

- **size** é a função primitiva do Clean que devolve o número de elementos de um vetor

- **x.[(size x) - 1]** pega o elemento cujo índice é o número de elementos do vetor (**size x**) menos 1 unidade.

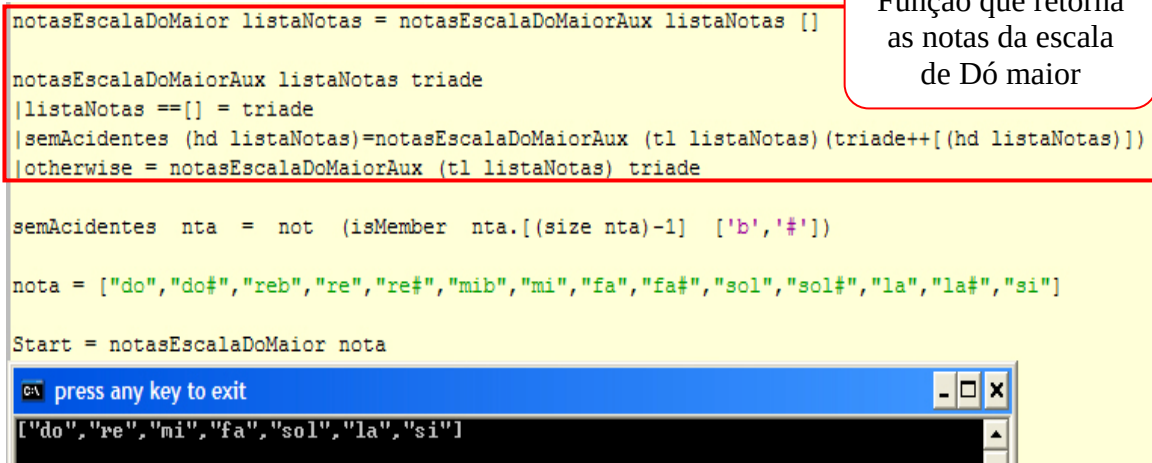
O primeiro elemento de um vetor é o de índice 0. Assim, (**size x**) -1 devolve o índice do último elemento do vetor x (o caractere que define o acidente).

programador que desconhece a linguagem Clean (previamente ciente dos símbolos utilizados: `\`, `<-:` e `|`).

Escala de dó maior = { `x \ x <-: Nm | not isMember x.[(size x)-1] ['b','#']` }

Onde: `Nm = {dó,dó#,réb,ré,ré#,mib,mi,fá,fá#,sol,sol#,lá,lá#,si}`

Para ilustrar quão aderente e legível é este código gerado para solucionar o problema proposto, observe, a seguir, como ficaria o mesmo programa empregando recursos comuns a paradigmas que não possuem notação Zermelo-Fraenkel. Nestes paradigmas, para encontrar o conjunto solução, dever-se-ia empregar uma estrutura recursiva com regras de parada e recursão para formação da lista das notas da escala de dó maior, como mostrado na listagem 3.1 a seguir:



Função que retorna as notas da escala de Dó maior

```
notasEscalaDoMaior listaNotas = notasEscalaDoMaiorAux listaNotas []

notasEscalaDoMaiorAux listaNotas triade
| listaNotas == [] = triade
| semAcidentes (hd listaNotas) = notasEscalaDoMaiorAux (tl listaNotas) (triade++[(hd listaNotas)])
| otherwise = notasEscalaDoMaiorAux (tl listaNotas) triade

semAcidentes nta = not (isMember nta.[(size nta)-1] ['b','#'])

nota = ["do","do#","reb","re","re#","mib","mi","fa","fa#","sol","sol#","la","la#","si"]

Start = notasEscalaDoMaior nota
```

press any key to exit

["do","re","mi","fa","sol","la","si"]

Listagem 3.1 – Função que retorna as notas da escala de dó maior.

Assim, se não fossem os nomes das funções auto-explicativas, a leitura do código não seria um processo tão simples para se saber o que o mesmo faz.

3.3 Aderência e Notação Zermelo-Fraenkel: Tríades musicais.

3.3.1 Tríades Maiores.

Como segundo exemplo, imagine agora que se deseja obter todas as tríades⁵⁷ que formam acordes maiores em um determinado *range* de notas musicais. Para tanto, deve-se, antes, conhecer o domínio de onde serão extraídas as tríades, bem como a regra de se criá-las.

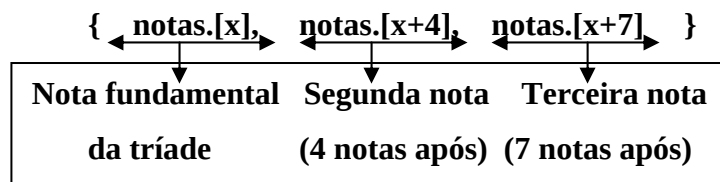
O domínio: É um conjunto de todas as notas musicais de onde se deseja extrair as tríades. Como exemplo, adota-se o domínio das notas musicais a escala cromática de **dó5** a **dó7**.

notas = {"do5", "do#5", "re5", "re#5", "mi5", "fa5", "fa#5", "sol5", "sol#5", "la5", "la#5", "si5", "do6", "do#6", "re6", "re#6", "mi6", "fa6", "fa#6", "sol6", "sol#6", "la6", "la#6", "si6", "do7"}

Regra: Uma tríade maior, como o próprio nome indica, deve possuir 3(três) notas musicais. Com licença a uma formalização musical adequada, segue-se a seguinte regra para uma tríade maior:

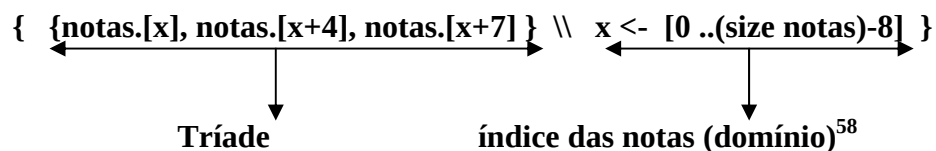
- A primeira é a nota fundamental da tríade;
- a segunda nota da tríade é a quarta nota após a fundamental;
- a terceira nota da tríade é a 7 nota após a fundamental.

Observe que a regra é criar, a partir do domínio, uma lista com três notas musicais, que seguem a lei de formação de uma tríade maior:



Desta forma, uma tríade maior partindo da nota “**dó5**” resultaria em: {"do5", "mi5", "sol5"}.

Implementação da função utilizando a notação Zermelo-Fraenkel:



⁵⁷ Grupo de três notas musicais

⁵⁸ Restringiu-se o domínio ao número das notas menos 8 * ((size notas)-8), devido cada tríade estar registrada em uma seqüência de 8(oito) notas musicais: A fundamental e a 7ª. nota após ela.

Ao executar a função, tem-se como reposta o conjunto solução, o conjunto imagem, contendo todas as tríades maiores no *range* fornecido (de **dó5** a **dó7**):

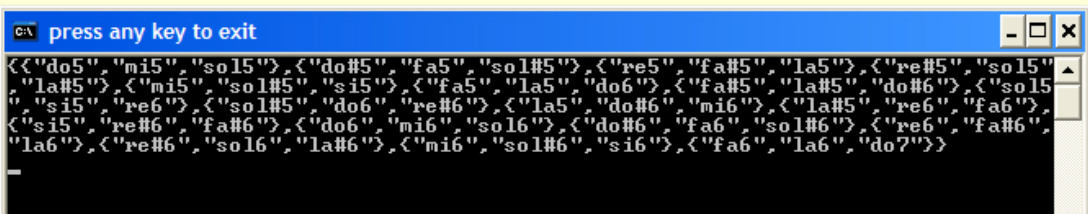
```
{{"do5","mi5","sol5"}, {"do#5","fa5","sol#5"}, {"re5","fa#5","la5"},
{"re#5","sol5","la#5"}, {"mi5","sol#5","si5"}, {"fa5","la5","do6"},
{"fa#5","la#5","do#6"}, {"sol5","si5","re6"}, {"sol#5","do6","re#6"},
{"la5","do#6","mi6"}, {"la#5","re6","fa6"}, {"si5","re#6","fa#6"},
{"do6","mi6","sol6"}, {"do#6","fa6","sol#6"}, {"re6","fa#6","la6"},
{"re#6","sol6","la#6"}, {"mi6","sol#6","si6"}, {"fa6","la6","do7"}}
```

Assim, ao observar o primeiro elemento do conjunto solução, tem-se: **do5**, **mi5**, **sol5**, ou seja, a primeira nota igual a **do5**, quatro notas depois da primeira (fundamental) tem-se o **mi5** e sete notas depois da fundamental tem-se a nota **sol5**, conforme lei de formação de um acorde maior. A próxima tríade começa na próxima nota fundamental, a nota **do#5**, e assim por diante. A última tríade completa possível, pela regra implementada, é **fa6**, **la6**, **do7**.

Observe, no programa implementado, que o mesmo para de procurar novas tríades quando faltarem 8 notas (`x <- [0 ..(size notas)-8]`). A listagem do programa e respectivo resultado são mostrados na listagem 3.2.

```
module triades
import StdEnv
notas::(String)
notas =("do5","do#5","re5","re#5","mi5","fa5","fa#5","sol5","sol#5","la5",
       "la#5","si5","do6","do#6","re6","re#6","mi6","fa6","fa#6","sol6",
       "sol#6","la6","la#6","si6","do7")

Start :: {(String)}
Start ={{notas.[x],notas.[x+4],notas.[x+7]}\ x <- [0 ..((size notas)-8)] }|
```



Listagem 3.2 – Listagem e resultado do programa triades

A legibilidade do código e a facilidade de reaproveitamento e de adaptação do mesmo fazem novamente do Clean uma linguagem bastante atraente.

3.3.2 Tríades menores.

Como exemplo de reaproveitamento de código, imagine que deseje tríades menores em vez de tríades maiores, ou que deseje implementar outro programa para tríades menores, utilizando o mesmo domínio. Neste caso, basta alterar no programa de tríades maiores a regra da função, adaptando-a para tríades menores. Uma tríade menor diferencia de uma tríade maior apenas na segunda nota da mesma, a segunda nota da tríade é a terceira nota após ela (na tríade maior era a quarta nota). Assim, basta modificar, na regra, a segunda nota da tríade, como segue;

- Regra da tríade maior: { notas.[x], notas.[x+4], notas.[x+7] }
- Regra da tríade menor: { notas.[x], notas.[x+3], notas.[x+7] }

Observe que tal modificação, dada neste exemplo, também seria simples de ser procedida em qualquer outra linguagem, mas, caso outra pessoa não fosse o programador original, tivesse que proceder tal modificação, em uma linguagem procedural, por exemplo, esta tarefa poderia ser mais trabalhosa do que em Clean. Diz-se isto devido ao fato de que nos programas implementados nestas linguagens terem várias estruturas do tipo FOR, IF e contadores que poderiam confundir o programador durante a análise do código, ou seja, o programa teria menos legibilidade para retrabalho e manutenção.

3.4 Implementações básicas.

O Anexo IX apresenta as principais ferramentas e técnicas de programação funcional utilizando a linguagem Clean, permitindo ao leitor que não está habituado com tal paradigma entender melhor os programas implementados nesta dissertação. No CD que acompanha este trabalho, foram colocados vários textos didáticos produzidos pela equipe do Laboratório de IA e Multimídia da FEELT-UFU, juntamente com exemplos e os programas utilizados nesta dissertação. Aconselha-se, para um leitor leigo nesta linguagem, consultar tal material para dirimir dúvidas e compreender algumas estruturas do Clean que não fiquem explicitamente claras nos desenvolvimentos que seguem.

3.4.1 Exemplo da função *StringTokens* aplicada ao domínio musical : Transpondo uma música em formato texto.

Como exemplo aplicado à música, foco de interesse deste trabalho, imagine ter um arquivo texto contendo uma música separada contendo, em cada linha, um compasso, tendo as notas musicais (da oitava 0 à oitava 9) com respectivas figuras, separadas por vírgula, obedecendo a seguinte sintaxe:

```

musica -> compasso maisCompassos
compasso -> notaFigura maisNotasFiguras
maisCompassos -> ("\\n")+compasso
notaFigura -> nota figura
maisNotasFiguras -> ",".notaFigura
nota -> (do|re|mi|fa|sol|la|si|do).(#[ε]).[0-9]
figura -> (ε)+.(b|sb|m|sm|c|sc|f|sf)

```

A figura 3.1 a seguir, mostra um arquivo texto com uma música na sintaxe descrita:

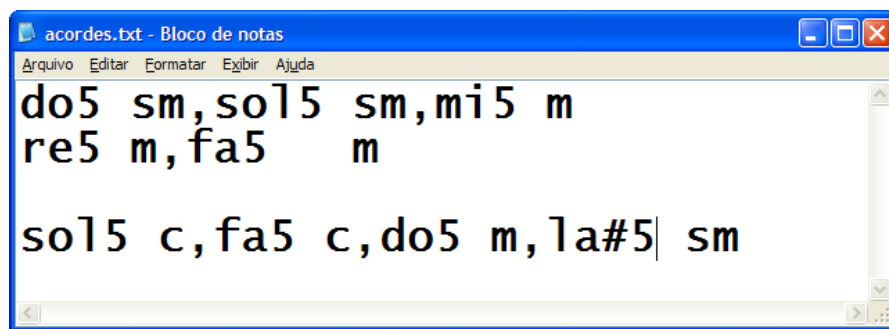


Figura 3.1 – Texto contendo uma música

A listagem 3.3 a seguir, mostra a lista de notas musicais implementadas, em Clean, segundo a regra: **nota -> (do|re|mi|fa|sol|la|si|do).(#[ε]).[0-9]**

```

33 notas :: [{Char}]
34 notas =
35 [ "do0", "do#0", "re0", "re#0", "mi0", "fa0", "fa#0", "sol0", "sol#0", "la0", "la#0", "si0",
36   "do1", "do#1", "re1", "re#1", "mi1", "fa1", "fa#1", "sol1", "sol#1", "la1", "la#1", "si1",
37   "do2", "do#2", "re2", "re#2", "mi2", "fa2", "fa#2", "sol2", "sol#2", "la2", "la#2", "si2",
38   "do3", "do#3", "re3", "re#3", "mi3", "fa3", "fa#3", "sol3", "sol#3", "la3", "la#3", "si3",
39   "do4", "do#4", "re4", "re#4", "mi4", "fa4", "fa#4", "sol4", "sol#4", "la4", "la#4", "si4",
40   "do5", "do#5", "re5", "re#5", "mi5", "fa5", "fa#5", "sol5", "sol#5", "la5", "la#5", "si5",
41   "do6", "do#6", "re6", "re#6", "mi6", "fa6", "fa#6", "sol6", "sol#6", "la6", "la#6", "si6",
42   "do7", "do#7", "re7", "re#7", "mi0", "fa7", "fa#7", "sol7", "sol#7", "la7", "la#7", "si7",
43   "do8", "do#8", "re8", "re#8", "mi8", "fa8", "fa#8", "sol8", "sol#8", "la8", "la#8", "si8",
44   "do9", "do#9", "re9", "re#9", "mi9", "fa9", "fa#9", "sol9", "sol#9", "la9", "la#9", "si9"]

```

Listagem 3.3 – Implementação de uma lista de notas musicais em Clean

Para consolidar o que já foi visto, apresentar-se-á, passo a passo, o processo de se ler o arquivo até a aplicação desejada que seja transpor as notas musicais conforme um determinado número de semitons. Nas implementações que seguem, utilizar-se-á a

biblioteca **StdArq**. Dentre as funções desta biblioteca, estão as funções de leitura de arquivo texto e a função de separar *tokens*.

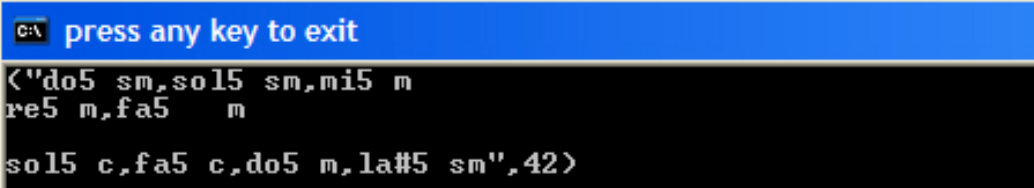
Primeiro passo: Ler o arquivo contendo a música em formato texto utilizando a função da **StdArq** *AbrirArquivoTextoExpl*.

```
6 #(conteudo, endereco, comp) = AbrirArquivoTextoExpl comp
```

Listagem 3.4 – Função abrir arquivo texto

A figura mostra o programa e a resposta contendo a *string* lida do arquivo “acordes.txt” mostrado na listagem 3.5.

```
1 module musicaTexto
2 import StdEnv, StdArq
3
4 Start :: *World -> ({#Char}, *World)
5 Start comp
6 #(conteudo, endereco, comp) = AbrirArquivoTextoExpl comp
7 =(conteudo, comp)
8
9
10
11
12
13
14
```



Listagem 3.5 – Lendo o arquivo contendo *string* com a música ser transposta

Observe o tipo da função **Start**:

```
Start :: *World -> ({#Char}, *World).
```

A mesma possui como argumento a variável do tipo ***World** (modificável) e devolve uma tupla com dois elementos: uma *String* (**{#Char}**) e a variável do tipo ***World** agora modificada devido à operação de IO de abrir e fechar um arquivo.

Segundo passo: Separar o texto, da *string* lida em compassos. Para tanto, utilizar-se-á a função **StringTokens** para separar os compassos.

```
7 #compassos_ = StringTokens isSepBarraN conteudo
```

Listagem 3.6 – Função *StringTokens*

O critério de separação é o caractere de saltar linha, testado pela função **isSepBarraN**.

```
13 isSepBarraN :: Char -> Bool
14 isSepBarraN x = '\n' == x
```

Listagem 3.7 – Função *isSepBarraN*

Esta função recebe como argumento um caractere (**Char**) e devolve um Booleano (**Bool**).

A listagem 3.8 mostra o programa de separar compassos.

```

1 module musicaTexto
2 import StdEnv, StdArq
3
4 Start :: *World -> ([{#Char}], *World)
5 Start comp
6 # (conteudo, endereco, comp) = AbrirArquivoTextoExpl comp
7 # compassos = StringTokens isSepBarraN conteudo
8 # = (compassos, comp)
9
10 isSepBarraN :: Char -> Bool
11 isSepBarraN x = '\n' == x
12
13
14 C:\ press any key to exit
15 <["do5 sm,sol5 sm,mi5 m","re5 m,fa5 m","sol5 c,fa5 c,do5 m,la#5 sm"],42>
16

```

Listagem 3.8 – Separando a música texto em uma lista de compassos

Observe agora, que a declaração de tipo da função **Start** muda, já que o resultado apresenta um tipo diferente da função anterior, ou seja, agora se tem uma lista de *strings*, em vez de *string*, como resultado da separação dos compassos.

Start :: *World -> ([{#Char}], *World),

onde cada *string* é um compasso da música. Observe, também, que foram eliminados os “saltar linha”, independente da quantidade deles entre linhas, conforme as regras de sintaxe propostas.

Observe que cada elemento da lista é uma *string* representando um compasso, no caso:

Compassos = [Compasso 1, Compasso 2, Compasso 3]

- **Compasso 1** = "do5 sm,sol5 sm,mi5 m"
- **Compasso 2** = "re5 m,fa5 m"
- **Compasso 3** = "sol5 c,fa5 c,do5 m,la#5 sm"

A função **StringTokens** aplica a função **isSepBarraN** em cada caractere da *string* contendo a música, criando uma lista de *strings* originalmente separadas pelo caractere de saltar linha ('\n').

Terceiro passo: De posse da lista de compassos, deve-se separar os conjuntos nota_figura existentes nos compassos já separados. Os conjuntos nota_figura estão separados em cada string_compasso pelo caractere vírgula (','). Assim, utiliza-se a função de teste deste caractere como argumento da função **StringTokens**, para separar estes conjuntos.

```

14 isSepVirgula :: Char -> Bool
15 isSepVirgula x = ',' == x

```

Listagem 3.9 – função isSepVirgula