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

Tradutor de Arquivos MIDI para Texto Utilizando Linguagem Funcional CLEAN

André Campos Machado

Uberlândia, outubro de 2001

Tradutor de Arquivos MIDI para Texto Utilizando Linguagem Funcional CLEAN

André Campos Machado

Dissertação apresentada como parte dos requisitos necessários para a obtenção do título de Mestre em Engenharia Elétrica da Universidade Federal de Uberlândia, sob a orientação do Prof. Dr. Luciano Vieira Lima.

Banca Examinadora:

Prof. PhD. Calimerio Augusto Soares Neto

Prof. PhD. Carlos Alberto Storti, – (UFU)

Prof. PhD. Keiji Yamanaka – (UFU)

Prof. Dr. Luciano Vieira Lima – Orientador (UFU)

Tradutor de Arquivos MIDI para Texto Utilizando Linguagem Funcional CLEAN

André Campos Machado

Dissertação apresentada como parte dos requisitos necessários para a obtenção do título de Mestre em Engenharia Elétrica da Universidade Federal de Uberlândia, sob a orientação do Prof. Dr. Luciano Vieira Lima.

Prof. Dr. Luciano Vieira Lima, Dr.
Orientador

Prof. Dr. Luis Carlos de Freitas
Coordenador da Pós-Graduação

À minha mãe Guaraciaba Silvia Campos e

à minha esposa Marília Mazzaro Pinto,

pelo incentivo e compreensão durante as horas dedicadas a este projeto.

Agradecimentos

Gostaria de expressar a minha gratidão ao Prof. Dr. Luciano Vieira Lima por sua orientação, encorajamento e amizade durante o desenvolvimento deste estudo e aos professores Dr. Calimerio Augusto Soares Neto, Dr. Carlos Alberto Storti, e Dr. Keiji Yamanaka que participaram da Banca de Defesa.

Ao Conservatório Estadual de Música Cora Pavan Capparelli de Uberlândia que em nome da Direção me forneceu o apoio necessário para a concretização do mestrado.

Aos meus alunos, incentivadores do meu desenvolvimento intelectual.

Ao Edson Simão, amigo e consultor nos momentos de dúvidas na editoração.

À todos aqueles que contribuíram de alguma forma, direta ou indiretamente, para a realização deste trabalho.

À Deus pela saúde necessária para a conclusão de mais uma etapa importante na minha vida.

Resumo

Um dos grandes desafios das atuais pesquisas envolvendo a manipulação de arquivos MIDI SMF, é a carência tanto de material didático específico para descrição do protocolo MIDI quanto de um sistema computacional baseado em uma linguagem aderente ao paradigma musical. Este trabalho foi desenvolvido com o objetivo de suprir esta deficiência, na medida em que cria ferramentas didáticas para manipulação de arquivos MIDI utilizando linguagem funcional CLEAN, a qual se mostra bastante aderente ao paradigma musical. Outro fator que determinou escolha por esta linguagem se deu pela velocidade e eficiência que a mesma possui na manipulação de listas, tanto ao utilizar as ferramentas básicas como, principalmente, ao utilizar a notação Zermelo-Frankel. O sistema, portanto, disponibiliza todas as ferramentas básicas de leitura dos arquivos MIDI padrões, e gera como produto final, um arquivo texto contendo todas as informações encontradas na música registrada.

Abstract

One of the greatest challenges of the present research involving the manipulation of MIDI SMF archives is the lack of specific instructional material, as in the description of the MIDI protocol, as for a computing system, which bears an adherent music language on the musical paradigm. The present work was developed with the objective to contribute to solve that need, creating instructing tools for the manipulation of MIDI archives using the a functional language CLEAN, which shows itself enough adherent on to the musical paradigm. Another choice factor for the present language took into account its great deal of speed and efficiency in the manipulation of the lists, as to the use of the main basic tools and as to the use of the Zermelo-Frankel notation. The system, makes available all the basic reading tools of the standard MIDI archives and produces, as a final product, a text archive which contains all information found in the recorded music.

Sumário

Capítulo 1 01

1.1. Estado da Arte 01

Capítulo 2 07

2.1. O arquivo WAVE 07

2.2. Arquivos no Formato WAVE 18

2.2.1. Taxa (freqüência) de amostragem 18

2.2.2. Número de canais 19

2.2.3. Quantização – resolução em bits 19

2.2.4. Tamanho do arquivo 21

2.3. Cabeçalho do arquivo WAVE 21

2.4. O Arquivo WAVE 21

Capítulo 3 24

3.1. O Nascimento do MIDI 24

3.2. MIDI e Computadores 26

3.3. O Que é MIDI 26

3.4. MIDI Ports 28

3.5. Conexão entre Dispositivos MIDI Utilizando os Ports MIDI IN, MIDI OUT e MIDI THRU 29

3.6. O Que Pode ser Feito com MIDI 34

3.7. Como são Gravadas as Músicas MIDI nos Teclados, Seqüenciadores e Computadores 34

3.8. Arquivos MIDI Standard MIDI File (SMF) 35
3.8.1. Entendendo como o MIDI funciona 36
3.8.2 Arquitetura da máquina MIDI 37
3.8.3. Representando uma nota musical e sua duração em Delta-Times 38
3.8.4. Pulses Per Quarter-note (ppq) 40
3.8.5. Delta Time 41
3.8.6. Analisando os arquivos MIDI SMF 45
3.8.7. Convertendo ppq para Delta Time 45
3.8.8. A diferença entre Formato 0 e Formato 1 46
3.9. O Arquivo MIDI SMF Formato 1 46
3.10. Meta-Eventos 48
3.10.1. Meta-Evento Set Tempo 48
3.10.2. Meta-Evento Fórmula de Compasso (Time Signature) 48
3.10.3. Meta-Evento Armadura de Clave (Key Signature) 49
3.10.4. Exemplo de arquivo MIDI SMF – Formato 1 50
3.10.5. Exemplo de arquivo MIDI SMF – Formato 1 – com Runing Status 52
Capítulo 4 55
4.1. Conceitos e Notação Musical 55
4.2. Notas Musicais 55
4.3. Figuras Musicais 57
4.4. Acidentes Musicais 58
4.5. Claves Musicais 59
4.5.1. Distribuição das Notas na Clave de Sol e de Fá 59
4.6. Fórmula de Compasso (Time Signature) 60
4.6.1. Exemplos de Compassos Simples 60
4.6.2. Exemplos de Compassos Compostos 61
4.7. Armadura de Clave (Key Signature) 61

4.8. Tonalidade Maior 62
4.9. Acorde 63
4.9.1. Cifras 64
4.9.2. Exemplos 64
4.9.3. Outras abreviações 64
4.10. Intervalos 65
4.10.1. Simples 65
4.10.2. Composto 65
4.10.3. Ascendente 65
4.10.4. Descendente 65
4.10.5. Melódico 65
4.10.6. Harmônico 65
4.10.7. Classificação dos intervalos usando como referência a escala de Dó Maior 66
4.10.8. Abreviações 66
4.10.9. Resumindo 66
4.11. Tonalidades Menores 66
4.11.1. Escala Menor Natural 66
4.11.2. Escala Menor Harmônica 67
4.11.3. Escala Menor Melódica 67
4.11.4. Os acordes de quatro notas montados sobre as notas das escalas 67
Capítulo 5 69
5.1. Projeto e implementação das rotinas (ferramentas) de manipulação de arquivos MIDI SMF formato 1 sem running status 69
5.1.1. Leitura do arquivo MIDI 70
5.1.1.1. Ler o arquivo MIDI e convertê-lo em uma lista de inteiros 71

5.1.1.2. Extrair o track principal do arquivo MIDI 72
5.1.1.3. Leitura do formato do arquivo SMF 73
5.1.1.4. Leitura do número de tracks 73
5.1.1.5. Leitura do valor da ppq 74
5.1.2. Ferramentas básicas gerais 75
5.1.2.1. Criando uma lista dos mTracks 76
5.1.2.2. Gerando uma lista de tracks 77
5.1.2.2.1. Pegando o conteúdo de um track 78
5.1.2.3. Separando todos os tracks de um arquivo MIDI 79
5.1.2.4. Eliminando o cabeçalho e número de bytes de cada track mTrk 80
5.1.2.5. Separando os eventos dos tracks 82
5.1.2.5.1. Primeiro Track 82
5.1.2.5.2. Convertendo Delta Times para ppq 83
5.1.2.5.2.1. Primeiro byte do Delta Time menor que 80_H (128_{10}) 83
5.1.2.5.2.2. Primeiro byte do delta time maior que 7F (127_{10}) e o segundo menor ou igual a 7F (127_{10}) 84
5.1.2.5.3. Separando os eventos dos demais tracks 88
5.1.2.6. Conversão de ppq em Delta Time 100
5.1.2.7. Metrônomo 100
5.1.2.8. Tonalidade da música 110
5.1.2.9. Fórmula de compasso 122
5.1.2.10. Instrumento do canal 133
5.1.3. Leitura dos tracks musicais e conversão dos mesmos em formato texto 146
Capítulo 6 173
6.1. Conclusões 173

6.2. Trabalhos Futuros 174
Anexos 175
A.1. MIDI Biblioteca.icl 175
A.2. MIDI Biblioteca.dcl 189
A.3. Sistemas de Numeração – Uma Breve Abordagem 188
A.3.1. Conversão Entre o Sistema Decimal e Outros Sistemas de Numeração 188
A.3.1.1. Base 10 188
A.3.1.2. Base 2 189
A.3.1.3. Base N 190
A.3.1.4. Base 16 190
Bibliografia 191

Capítulo 1

1.1. Estado da Arte

O desenvolvimento tecnológico causou grande impacto em várias áreas do conhecimento, forçando o homem a procurar um equilíbrio entre o que a ciência oferece e suas reais necessidades. A função do computador neste contexto é servir de ferramenta para desempenhar as várias funções dele exigidas.

Até o advento da Música Eletrônica, na metade do século XX [48], os instrumentos musicais passaram a se desenvolver não só mais pela necessidade musical, mas também visando o aproveitamento das novas possibilidades tecnológicas. Na década de 50, o americano Bob Moog [3] [40] criou o sintetizador, um instrumento eletrônico controlado por um teclado, que era capaz de sintetizar sons de outros instrumentos. Esta criação foi o grande impulso necessário para o desenvolvimento da tecnologia musical e, principalmente, do protocolo **MIDI**[1].

Novos recursos foram inseridos em antigos instrumentos, bem como foram desenvolvidos novos para atender às necessidades evolutivas da música. Um exemplo claro de evolução instrumental é o Piano, que veio atender a uma necessidade crescente de expressão musical, na medida em que possibilitava contrastes de timbre e de dinâmica [58], recursos que não eram possíveis até então no Cravo.

Os primeiros praticantes da Música Eletrônica tiveram que enfrentar dois grandes problemas [47]:

---

[1] **M**usical **I**nstrument **D**igital **I**nterface - Interface Digital para Instrumentos Musicais

- equipamentos primitivos, que tornavam quase que impossível a criação de sons semelhantes à música tradicional;
- a falta de conhecimento das características físicas e acústicas do som, conteúdo que não faz parte de um treinamento musical tradicional.

Com a invenção do transistor na década de 40 [52], permitiu-se a criação de computadores menores, mais sofisticados e que puderam também, mais tarde, ser utilizados para produzir sons musicais. Em 1957, Max V. Matthews, que na época trabalhava na AT&T, tornou-se a primeira pessoa a programar um computador digital com o objetivo de produzir música [52]. Não foi, a princípio, uma música de grande qualidade mas com o desenvolvimento da tecnologia e principalmente dos computadores isto logo seria resolvido, ou pelo menos, amenizado. A partir de então, os teatros passaram a incorporar não somente instrumentos de sopro, cordas e percussão mas, também, alto-falantes, aparelhos eletrônicos e instrumentos sintetizadores.

A utilização do computador na música facilita a integração do músico, em suas várias especialidades, com os conhecimentos tradicionais e as novas tecnologias emergentes. A utilização do computador economiza anos de treinamento quando dotado de ferramentas computacionais que automatizam atitudes meramente mecânicas e repetitivas, permitindo ao músico se especializar mais eficientemente no domínio escolhido, seja como professor, intérprete, arranjador ou mesmo como compositor [55].

As novas tecnologias propiciaram o surgimento de vários sintetizadores timbrais[2], e, com eles, surgiram novos problemas os quais passaram exigir novas soluções. A partir deste momento, os músicos começaram a utilizar em seus shows vários sintetizadores inicialmente monofônicos[3] e monotimbrais[4], gerando a

[2] Aparelho capaz de sintetizar timbres de instrumentos musicais.

[3] Capacidade de produzir apenas uma nota musical de cada vez.

[4] Capacidade de produzir apenas um timbre, ou seja, apenas um instrumento musical.

necessidade da interligação entre eles para produzir o efeito de polifonia e multitimbralidade [40] [53].

Inicialmente, cada fabricante produziu seu próprio protocolo para interligação de seus produtos, porém, os músicos nunca se contentaram em comprar equipamentos de apenas um fabricante. O grande problema nesta fase foi o fato de que os protocolos de interligação eram incompatíveis entre as diversas marcas, gerando a necessidade de um protocolo único. Para resolver este problema de comunicação entre os sintetizadores de fabricantes diferentes, surgiu então, o protocolo MIDI, onde todos os fabricantes passaram a incluir nos processadores de seus sintetizadores, além dos protocolos específicos, também, o protocolo MIDI [40] [52]. A partir deste momento, abriu-se um novo horizonte para o desenvolvimento da tecnologia musical, pois, além de permitir a comunicação entre teclados, o padrão MIDI passou a ser utilizado por vários outros equipamentos, tais como: computadores, jogos, programas de multimídia e, também, como alternativa para transmissão de arquivos musicais pela internet, já que o arquivo MIDI é um arquivo bastante compacto.

Trabalhar com o protocolo MIDI em computadores é algo comum nos dias de hoje em vista da extensa aplicabilidade comercial e de entretenimento. Os arquivos MIDI SMF[5] vem ganhando cada vez mais espaço em aplicativos multimídia, principalmente devido ao fato de ser um arquivo altamente portável e por permitir a armazenagem de vários minutos de música em poucos Kbytes [55].

O fato do MIDI ser um protocolo de comunicação e não um arquivo de música digitalizada, ou seja, o mesmo apenas armazena os comandos que deverão ser seguidos por um processador para que uma música seja gerada em um equipamento MIDI dotado de módulo timbral[6], faz-se necessário, portanto, que o usuário do mesmo possua tal módulo [40] [53].

O grande problema da popularização do MIDI em computadores foi o custo elevado dos módulos e placas de som que reproduzissem os timbres dos instrumentos

---

[5] SMF: Standard MIDI File – Arquivos MIDI padrões.

[6] O módulo timbral é normalmente denominado pelos profissionais da computação por placa de som

acústicos com fidelidade. Inicialmente, nem mesmos os módulos caros conseguiram realizar sínteses satisfatórias. Mesmo assim, vários fabricantes de placas de computadores apostaram no protocolo, e disponibilizaram em suas placas de som chips de síntese de instrumentos musicais, baseados no protocolo MIDI, que foram inicialmente utilizados em jogos de computadores [52].

Devido ao alto grau de compactação das informações musicais capazes de reproduzir músicas, trilhas e efeitos sonoros com qualidade, mesmo a um custo relativamente elevado para usuários de entretenimento, grandes empresas de *software* passaram a desenvolver sistemas de síntese em tempo real que simulassem os bons resultados obtidos pelos módulos de qualidade reconhecida. Com o avanço da tecnologia na produção de máquinas velozes e memórias de alta capacidade de armazenamento de dados, houve um crescente surgimento de vários programas dedicados à automação dos domínios musicais, propiciando a interação do músico com o computador, através de uma linguagem que ele já conhece, como a partitura, os controles de um gravador e de mesa de mixagem.

A partir deste momento, surgiu a necessidade de especialistas que entendessem e conseguissem manipular o protocolo MIDI em tempo real, tanto para controle quanto para síntese.

Um dos pontos que limitam a criação de novos programas por qualquer bom programador, está na carência de material informativo, principalmente didático, sobre a forma de manipulação e criação de arquivos SMF. Mesmo com toda tecnologia atual, os programas existentes para este domínio ainda possuem limitações oriundas, claramente, da falta de informação adequada. Assim, apesar de existirem vários textos na internet sobre o tema, e, também, em alguns livros [30] [39] [40] [52] [53], os mesmos não são suficientes para permitir a formação adequada do profissional exigido nesta área pelo mercado de trabalho atual. A prova disto é que artigos apenas conceituais sobre o tema são normalmente aceitos por congressos e revistas que consagradamente só aceitam artigos inovadores. [22]

Um outro problema que agrava a falta de profissionais com tal embasamento é que a maioria dos sistemas de síntese, edição e transmissão do protocolo MIDI, estão

escritos em linguagens procedurais, reconhecidamente sujeitas a efeitos colaterais imprevisíveis os quais tornam os sistemas mais lentos e sujeitos a falhas. O ideal seria que a comunidade científica disponibilizasse às empresas, profissionais com embasamento MIDI adequado e com a proficiência em programação correta utilizando linguagens compiladas e que não possuam efeitos colaterais, tais como as linguagens com paradigma Funcional. Neste ponto, um novo problema é acrescido ao primeiro, onde bons programadores em linguagens funcionais são tão raros quanto bons conhecedores do protocolo MIDI. Mais raro ainda é encontrar um profissional com as duas habilitações, já que as mesmas são, atualmente, diametralmente opostas em foco e aplicação.

Assim, dentre tantos trabalhos interessantes e emergentes que o grupo da Computação Sônica vem demandando das empresas e grupos de pesquisas, objetiva-se neste trabalho produzir um texto que permita sanar a necessidade mais básica e angular de todas as aplicações, ou seja, disponibilizar a conceituação didática do protocolo MIDI e dos arquivos por ele manipulados, utilizando como linguagem de implementação das rotinas básicas uma que possua o paradigma funcional, com clareza e aderência aos conceitos apresentados nesta dissertação. A escolha pela linguagem CLEAN se deu devido à sua forte aderência ao paradigma escolhido, por ser veloz e pelas ferramentas computacionais que a mesma disponibiliza ao programador no tratamento de listas, tais como, notação Zermelo-Frankel [24], e por sua eficiência na manipulação deste tipo de dados, os quais serão fortemente manipulados na implementação das rotinas MIDI.

O objetivo, portanto, deste trabalho é produzir um texto e programas aplicativos didáticos que permitam a um usuário de computação entender o protocolo MIDI e saber implementar rotinas em linguagem funcional para extração dos eventos musicais contidos nos mesmos.

Assim, justifica-se este trabalho pela carência de material didático e pela necessidade constante dos meios de comunicação que utilizam multimídia. Grandes investimentos estão sendo aplicados em desenvolvedores de sistema para internet os quais necessitam, cada vez mais, de maior velocidade na transmissão de músicas em

tempo real. Estes problemas serão solucionados na medida que mais pessoas passem a ter tecnologia e informação para trabalhar os diversos protocolos existentes. Após experimentar vários formatos de arquivos, o mp3 e MIDI têm sido adotados como a mídia de foco atual para estes casos, em detrimento dos arquivos WAV, Au, e outros mais, que ocupam grande volume de memória inviabilizando sua utilização nos atuais e lentos sistemas da rede mundial de comunicação.

Capítulo 2

2.1. O arquivo WAVE

A maioria dos profissionais em gravação de áudio digital, que manipulam essencialmente arquivos WAVE, tem certa dificuldade para entender os processos de gravação e edição de arquivos MIDI. O que ocorre é que os dois sistemas de representações digitais para música apresentam uma filosofia metodológica de implementação totalmente diferentes entre si [40].

No arquivo WAVE, o áudio é digitalizado, onde cada ponto do sinal sonoro é amostrado e quantizado obtendo-se, assim, um arquivo digital que discretiza o som original sem perda significativa da qualidade sonora durante a reprodução dos mesmos. Desta forma, a qualidade obtida na discretização do sinal de áudio dependerá do conversor A/D utilizado e da taxa de amostragem dos dados, obedecendo o critério definido por Nyquist[7] [12], que estabelece: a taxa de amostragem deverá ser o dobro da maior freqüência significativa do sinal a ser reproduzido. Devido ao tipo de aquisição, o sinal de áudio ocupa bastante espaço na hora de armazena-lo no computador.

O arquivo MIDI é um protocolo de comunicação padrão entre dispositivos eletrônicos, os quais podem trocar mensagens de como a música deve ser reproduzida pelos seqüenciadores e módulos de timbres [3]. O arquivo MIDI, portanto, não armazena uma cópia do sinal de áudio digitalizado, mas apenas as informações de como um dispositivo MIDI dotado de um módulo de timbres deverá executar uma determinada música. Desta forma, o responsável pela qualidade do timbre do

[7] Nyquist provou que a taxa de amostragem de um sinal deverá ser duas vezes o valor da maior freqüência que se deseja reproduzir.

instrumento reproduzido é o módulo timbral e não do arquivo MIDI gravado. Uma placa de som tipo Sound Blaster [53] ou compatível possui internamente um módulo timbral ou um sistema de reprodução de timbres armazenada em memória. Módulos Timbrais [53] de fabricantes diferentes reproduzirão a música gravada com a mesma dinâmica, mas com timbres diferentes.[8]

Além da significativa redução do espaço de memória necessário para registrar a mesma música, existem outras vantagens de se trabalhar com MIDI e não com arquivos WAVE. Uma das principais, do ponto de vista de um músico, é que o protocolo do arquivo MIDI traz consigo todas as informações do domínio musical necessárias para um músico ou programa computacional realizar futuras análises da música em questão. Dentre estas informações pode-se citar: fórmula de compasso, unidade de tempo, armadura de clave, andamento, instrumento, as notas musicais existentes, lirismo, nome da música e outras informações relevantes[9].

Os problemas de se construir sistemas computacionais dedicados à análise e reconhecimento dos eventos músicas, baseando-se em arquivos de áudio são, basicamente, os seguintes:

- Como separar os instrumentos e seus respectivos eventos, já que os mesmos estão registrados em uma única forma de onda?
- Caso fosse possível separar os instrumentos, como extrair os parâmetros musicais, já que estes não estão disponibilizados explicitamente na música?

Isto, sem dúvida alguma, é uma tarefa ainda difícil e desacreditada por qualquer profissional que reconheça as dificuldades a serem enfrentadas.

Os exemplos a seguir visam tornar claro as dificuldades citadas anteriormente. Para tanto, serão mostrados arquivos sonoros em formato WAVE, contendo ora sinal de uma nota musical tocada no violão, ora uma nota musical tocada no piano e, concluindo, as duas notas simultaneamente. Ao analisar o exemplo das notas mixadas, o leitor poderá concluir o quão difícil é a tarefa de se reconhecer a existência de dois

[8] Como exemplo, o piano do módulo da Roland é diferente do piano do módulo da Yamaha, Korg, etc.
[9] Estes conceitos podem ser vistos no capítulo Conceitos e Notação Musical.

instrumentos diferentes tocando a mesma nota musical, já que os períodos das mesmas são idênticos. Ao mesmo tempo, a cada exemplo, pode-se observar a sensível redução do código gerado pelos arquivos MIDI.

Exemplo 1: Forma de onda da nota musical lá 5 em formato WAVE, tocada pelo instrumento piano.

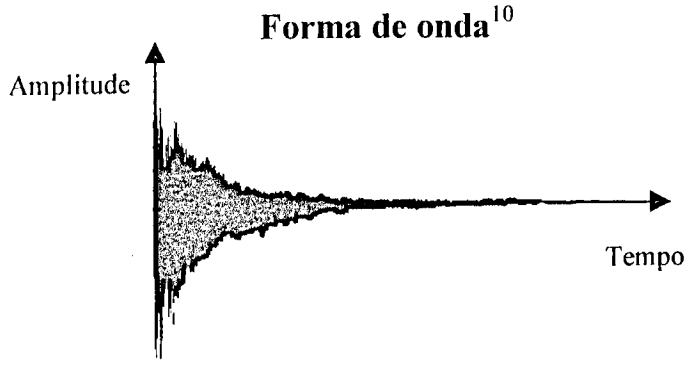


Figura 1

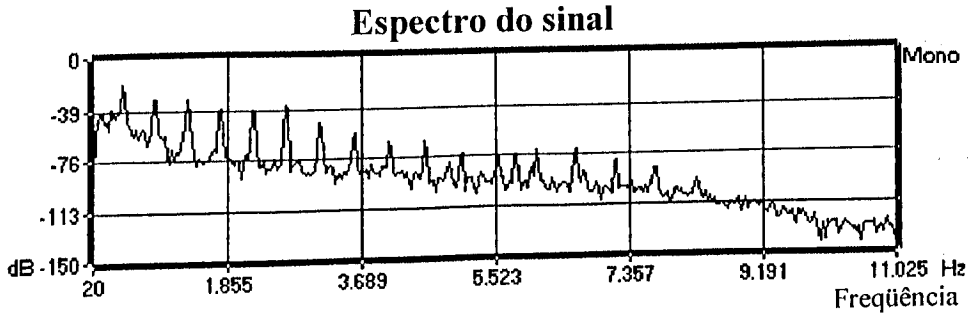


Figura 2

[10] O arquivo foi digitalizado no software Sound Forge 5.0 da Sonic Foundry.

Arquivo WAVE do sinal[11]

1/15 do arquivo WAVE gerado pela nota Lá 5

Hexadecimal ASCII

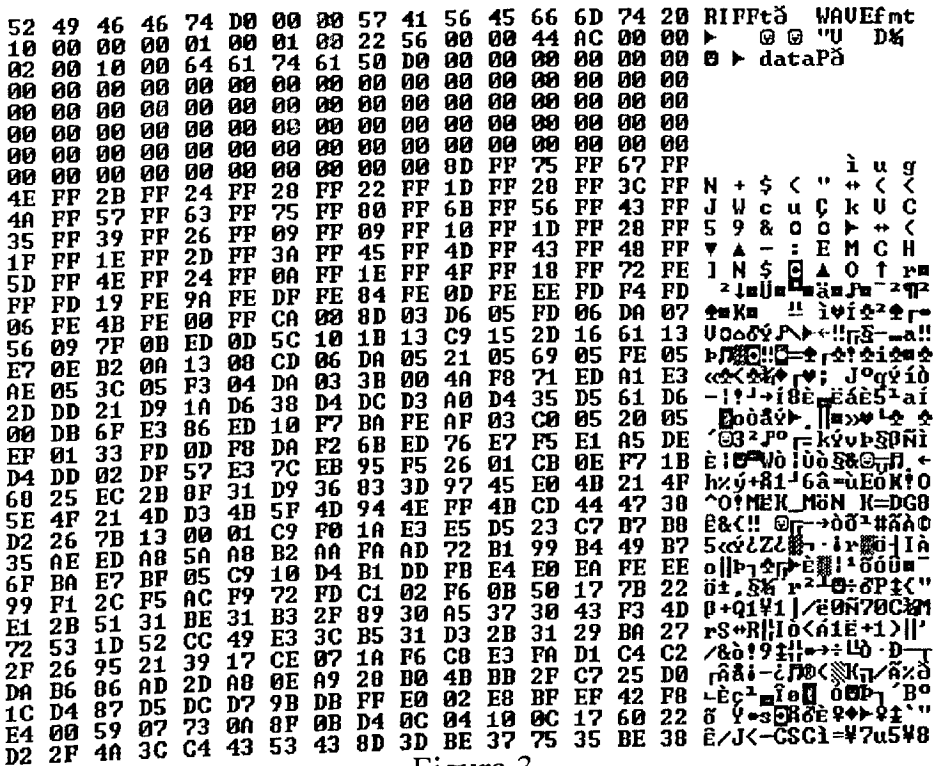


Figura 3

SMF do Piano tocando a nota Lá5 (Hexadecimal)

Hexadecimal ASCII

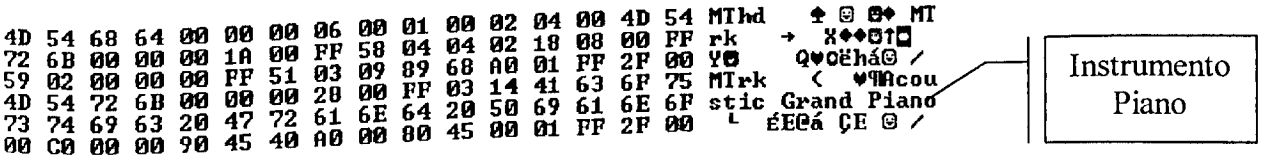


Figura 4

Ambos os arquivos deste exemplo retratam a partitura da figura 5. Ressalta-se que, no caso do arquivo WAVE, as informações notadas na mesma não estão presentes, e, no arquivo MIDI, estão, inclusive que o instrumento que está tocando é o piano (ver linha 5 da figura 4 no código ASCII: Grand Piano).

[11] O arquivo Hexadecimal foi gerado pelo software Direct Edit da Great Codeholio.

Partitura do Piano tocando a nota Lá5[12]

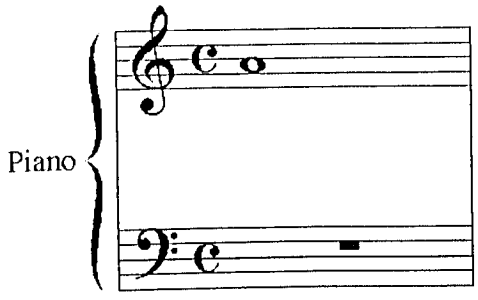


Figura 5

Exemplo 2: Forma de onda da nota musical lá 5 em formato WAVE, tocada pelo instrumento Violão.

Forma de onda

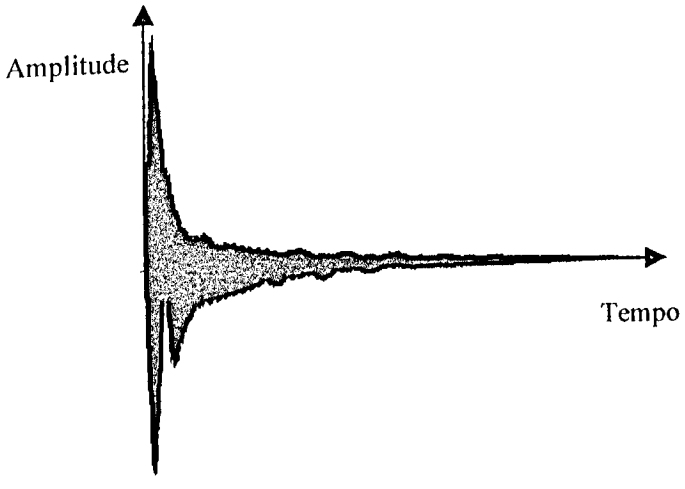


Figura 6

Espectro do sinal

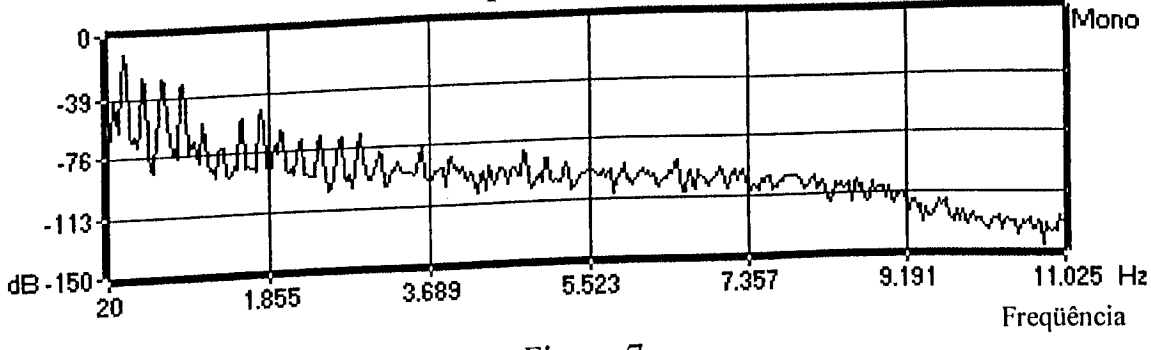


Figura 7

[12] A partitura foi gerada no software Finale 2001 da Coda Music Technology.

Arquivo WAVE do sinal

1/15 do arquivo WAVE gerado pela nota Lá 5

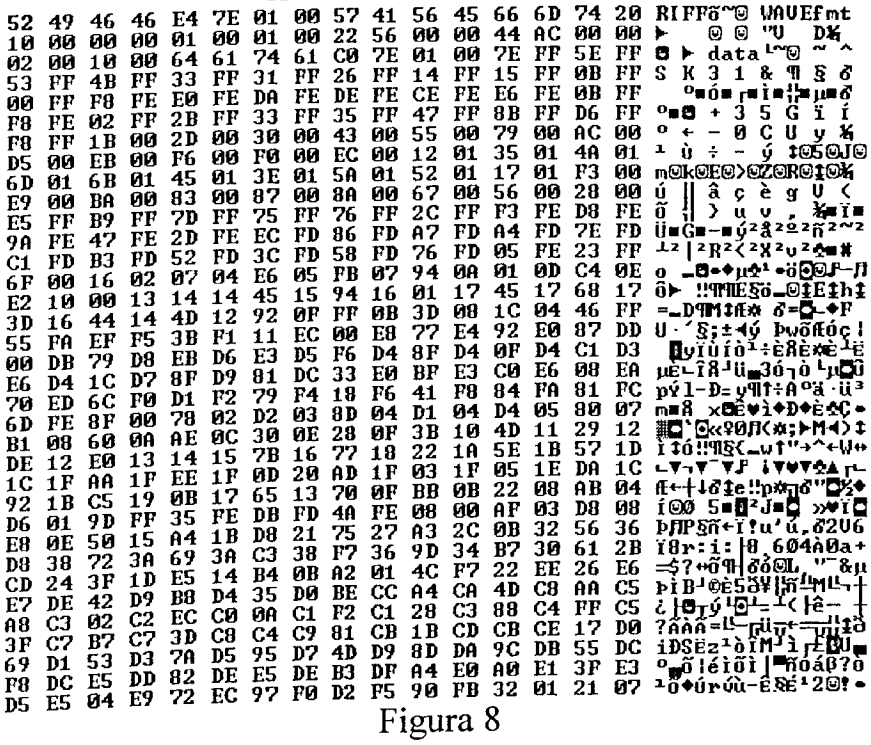


Figura 8

SMF do Violão tocando a nota Lá5

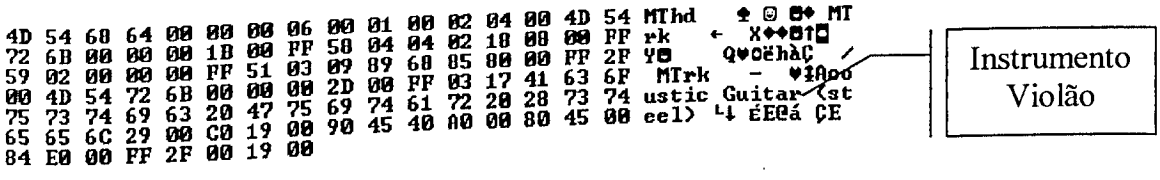


Figura 9

Ambos os arquivos deste exemplo retratam a partitura da figura 10.

Partitura do Violão tocando a nota Lá5[13]

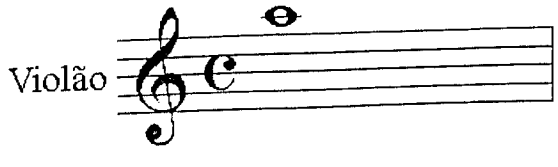


Figura 10

Exemplo 3: Forma de onda da nota musical lá 5 em formato WAVE, tocada simultaneamente pelos instrumentos piano e violão.

[13] O violão é um instrumento transpositor de oitava, portanto esta nota soa uma oitava abaixo.

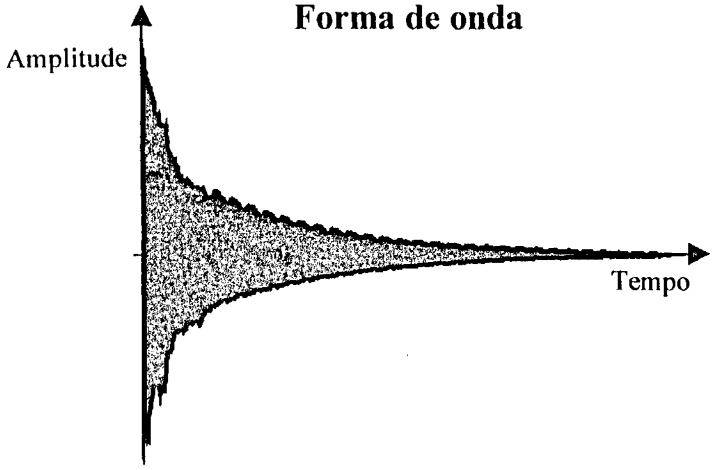


Figura 11

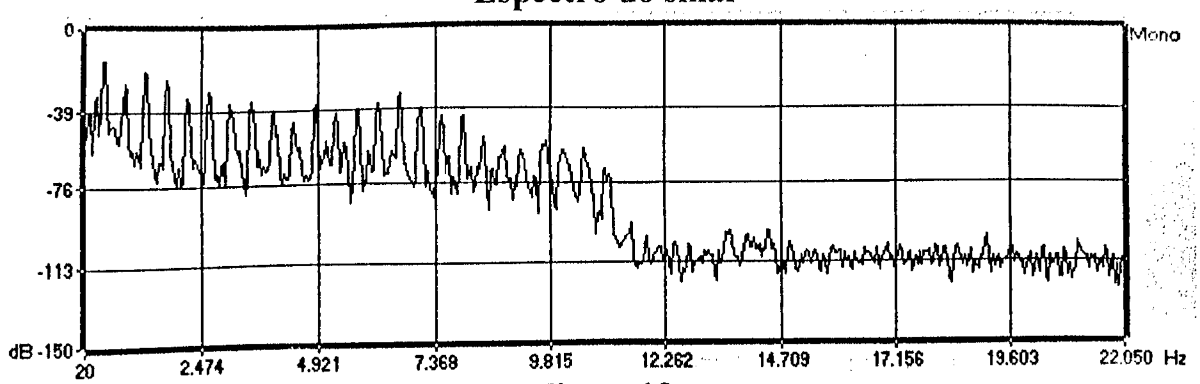


Figura 12

Freqüência

Arquivo WAVE do sinal

1/15 do arquivo WAVE gerado pela nota Lá 5

Hexadecimal — ASCII


```
52 49 46 46 AA DD 03 00 57 41 56 45 66 6D 74 20   RIFF WAVEfmt
10 00 00 00 01 00 01 00 44 AC 00 00 08 58 01 00
02 00 10 00 64 61 74 61 86 DD 03 00 6A FF CA FF   data
BB 00 60 00 B3 FE 43 FE EF FF 8D 00 96 FE 1D FE
FE 02 DC 08 B2 06 11 FC AC F4 52 FB 5D 0B 0F 14
C7 0A EE F6 CA E8 7B E6 F8 E6 26 E1 5B DA 8D DE
F5 EB C1 F0 9F E1 78 CB AE C3 AE CB EE D0 B2 C9
E9 C3 C8 D0 94 EA AD F9 CB F1 7C E0 44 DB 56 E6
D8 F3 C7 F7 B4 F4 1C F3 F5 F3 63 F1 CA EA BC E7
B5 EC FF F2 29 F1 68 E7 B8 DD B7 D8 B9 D7 F9 DA
D0 E3 7D ED 5B EF EE E7 67 E2 66 EA 41 FD 26 0D
6F 10 7A 09 42 02 F9 02 5C 0D 57 1D 17 2D D3 38
AB 3F B5 40 EC 3A 5F 30 DB 27 BA 27 0E 31 DF 40
A9 53 EE 63 B1 68 D2 5B 2D 43 57 2E 41 27 53 2A
BF 2E EF 31 E0 36 7D 3D D8 41 40 42 7D 41 89 41
13 40 3F 3B 09 35 D4 2E 7F 25 B7 15 06 03 D7 F5
8B F2 BA F5 B5 FB AE 06 6A 1A 08 32 4D 40 43 3B
13 27 04 13 2D 0C 9E 13 4A 1F F6 22 7A 18 C5 03
5D F0 41 E8 6C EB 8F F0 55 EE 77 E2 8A D0 C2 BC
2A AA 36 9B 8C 90 7E 89 10 88 47 91 19 A6 62 BE
4C CE A5 D1 EA CE 8A D0 44 DD D1 F4 08 0E E2 1C
47 17 04 02 49 EA 6A DB EB D9 28 E4 C7 F1 09 F4
91 E0 4F BF D3 A5 C9 A0 FC A7 06 AB AF A4 01 9C
B3 97 79 97 07 90 29 99 1D 97 05 91 16 8D AB 8D
C7 9D 00 AE D2 B0 BE B6 0A AD 62 A5 2C A7 86 D3
2D C5 17 D5 88 E0 E3 EA 5C F8 04 06 B6 0A F5 00
F3 EE 7E E1 F7 DF 88 E6 53 ED 04 F1 22 F4 69 FA
4B 04 6F 0E B3 13 8E 11 CB 0A A6 04 AD FF F6 F6
3A E9 99 DE 24 E0 32 EB 80 F3 7E F2 33 F0 2A FA
B9 14 48 37 D7 53 5B 60 1C 5C 54 50 BF 49 FD 4D
8F 57 74 5C A2 58 43 50 D7 49 10 49 52 4D 15 52
80 4F 06 40 8E 26 9D 0E 5E 01 BA FB 33 F2 A3 DF
77 CD F9 C9 CE D9 95 F5 2A 13 FC 2B 3B 3B 1D 3F
F2 3D 0C 45 CB 5A 59 74 FF 7F CF 77 47 65 D3 54
46 4B 1E 40 53 47 5D 42 0F 33 69 1E 54 0F CF 0E
```

Figura 13

SMF do Violão e Piano tocando a nota Lá5

```
4D 54 68 64 00 00 00 06 00 01 00 03 04 00 4D 54 MThd    ♠ ⊖ ♥♦ MT
72 6B 00 00 00 1A 00 FF 58 04 04 02 18 08 00 FF rk   →  X♦♦☺↑◘
59 02 00 00 00 FF 51 03 09 89 68 A0 01 FF 2F 00 Y☺    Q♥cëhá☺ /
4D 54 72 6B 00 00 00 2B 00 FF 03 17 41 63 6F 75 MTrk    +  ♥↕☺cou
73 74 69 63 20 47 75 69 74 61 72 20 28 73 74 65 stic Guitar (ste
65 6C 29 00 C0 19 00 90 45 40 A0 00 80 45 00 01 el) └↓ ÉE@á ÇE ☺
FF 2F 00 4D 54 72 6B 00 00 00 28 00 FF 03 14 41  / MTrk   (  ♥¶A
63 6F 75 73 74 69 63 20 47 72 61 6E 64 20 50 69 coustic Grand Pi
61 6E 6F 00 C1 00 00 91 45 40 A0 00 81 45 00 01 ano ┴  æE@á üE ☺
FF 2F 00 00
```

Instrumento Violão

Instrumento Piano

Figura 14

Obs.: Ambos os arquivos deste exemplo retratam a partitura da figura 15.

Partitura do Violão e Piano tocando a nota Lá5

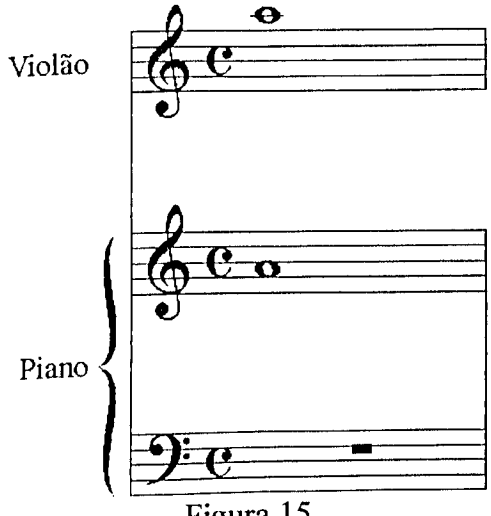


Figura 15

O exemplo 3 ilustra a forma de onda da nota lá 5, executada simultaneamente pelo piano e o violão, com seu respectivo espectro. A simples análise espectral dos sinais não permitem reconhecer, com segurança, qual instrumento está tocando, mas apenas que a nota lá 5 está sendo executada, não sendo possível quantizar por quantos instrumentos. O estado atual da arte se foca, ainda que com um reconhecimento mais básico, ou seja, reconhecer a freqüência da nota que está tocando[14], já que, mesmo afinados, os instrumentos acústicos não tocam exatamente a mesma freqüência.

Já existem sistemas razoáveis de conversão áudio-MIDI, mas o que ocorre é que os mesmos não são eficientes e, o pouco que acertam, não possuem nenhum compromisso timbral[15]. Reconhecer mesmo apenas as notas musicais de um único

[14] Autoscore, Finale, Sibelius, Wave to MIDI.
[15] Não se preocupam em definir o instrumento.

instrumento musical é uma tarefa cujas expectativas ainda estão sendo depositadas em teses de doutorado e grupos de pesquisas, como é o caso da UFU, que prometem para 2002 uma solução polifônica e monotimbral [21]. A solução de interpretação de uma música através de um arquivo de áudio é algo bem complexo, principalmente devido ao fato de não se conhecer com precisão as notas musicais que lá estão representadas, muito menos seus timbres, ou seja, seus instrumentos.

A aquisição da melodia através de uma partitura é uma tarefa bem mais plausível, ou seja, mais fácil de ser realizada, apesar de inexistente. Diz-se mais fácil devido ao fato de já se ter, na partitura, os instrumentos (timbres) separados um a um, com a informação das notas tocadas por cada um e a separação dos compassos em tempos corretos. Porém, esta solução exige que um músico experiente extraia as informações necessárias da mesma e as converta para um formato que o computador consiga interpretar corretamente. O exemplo a seguir ilustra o que foi dito, onde é apresentado um trecho da música *"De mais ninguém"* de Marisa Monte e Arnaldo Antunes, onde além de vários instrumentos, aparece, também, a partitura do cantor com a letra, onde, um músico, mesmo sem grandes qualificações, pode extrair as informações necessárias à análise musical. Já, em um arquivo sonoro da mesma música, nem sempre uma pessoa com bom ouvido e audição conseguem sequer entender o que o cantor está cantando.

Partitura de um trecho da música De mais ninguém

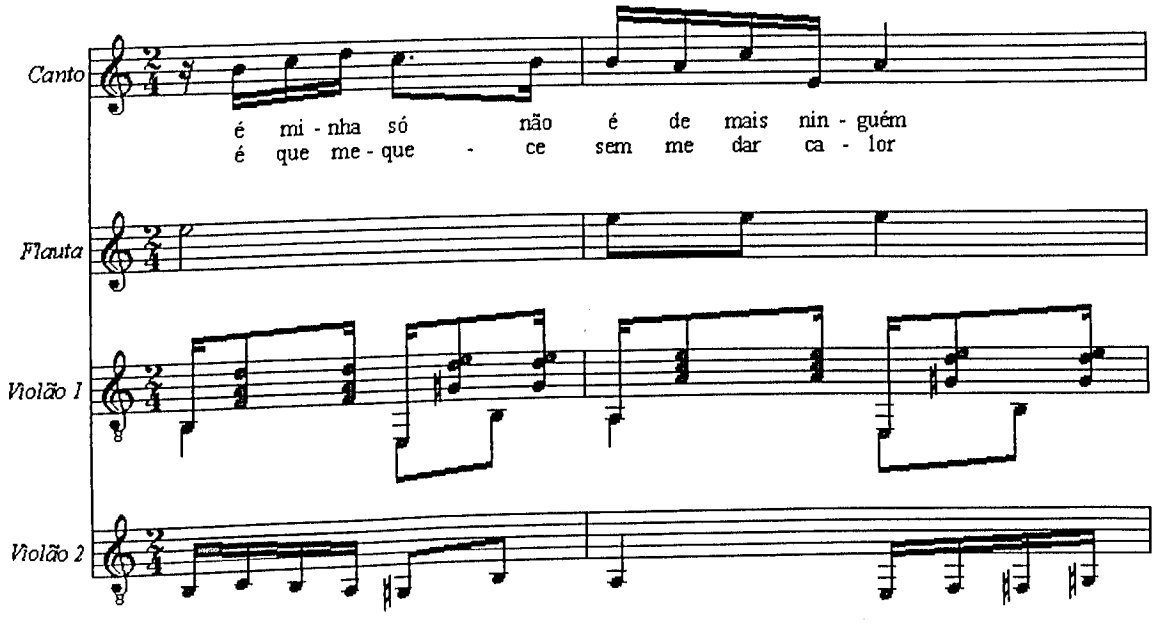


Figura 16

Esta mesma partitura é representada a seguir em arquivo Wave, onde a análise espectral e o arquivo hexadecimal da mesma pouco ou nada contribui para o reconhecimento dos instrumentos, da melodia e da letra da música.

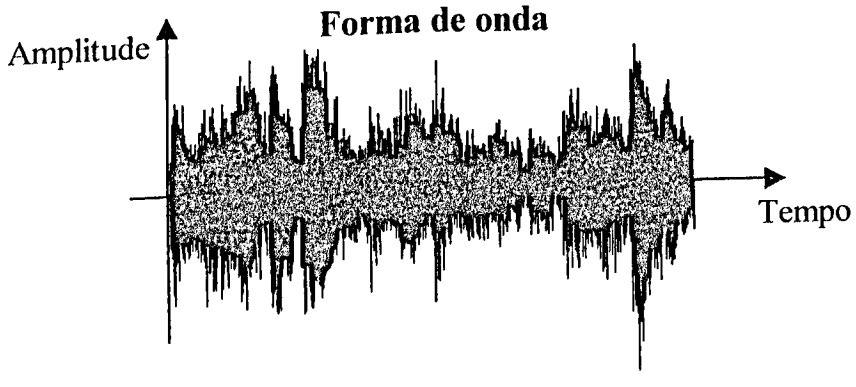


Figura 17

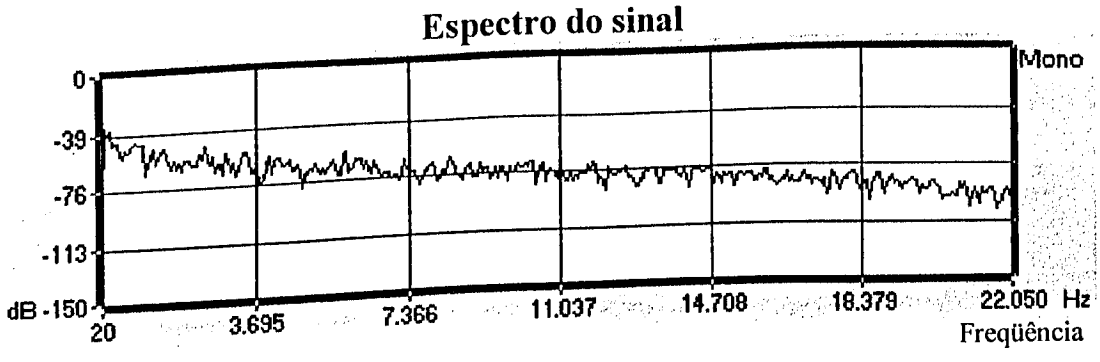


Figura 18

Arquivo WAVE do sinal

1/15 do arquivo WAVE de dois compassos da música

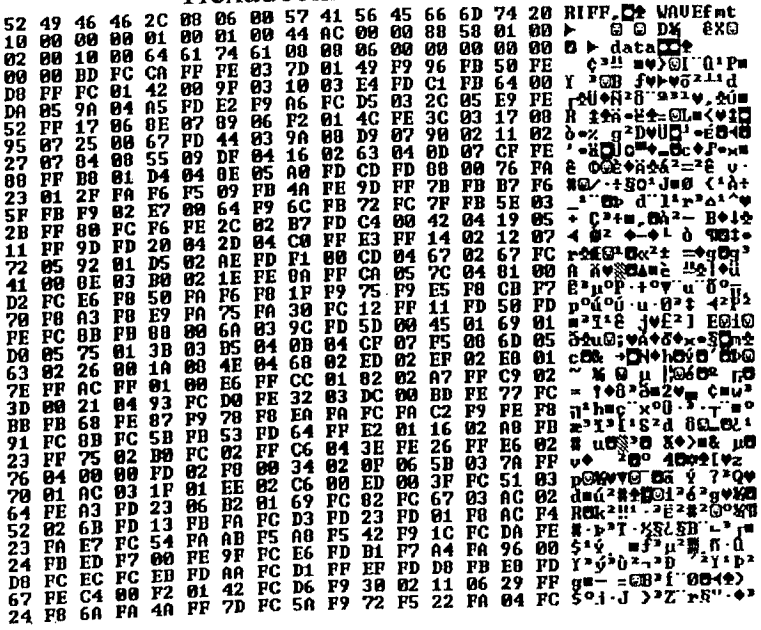


Figura 19

O arquivo MIDI é a melhor solução intermediária entre o conhecimento musical e um modelo computacional mais aderente ao programador. No mesmo tem-se registrado todas as informações desejadas, de forma semelhante à notada em uma partitura. Veja esta mesma música em arquivo MIDI padrão (SMF):

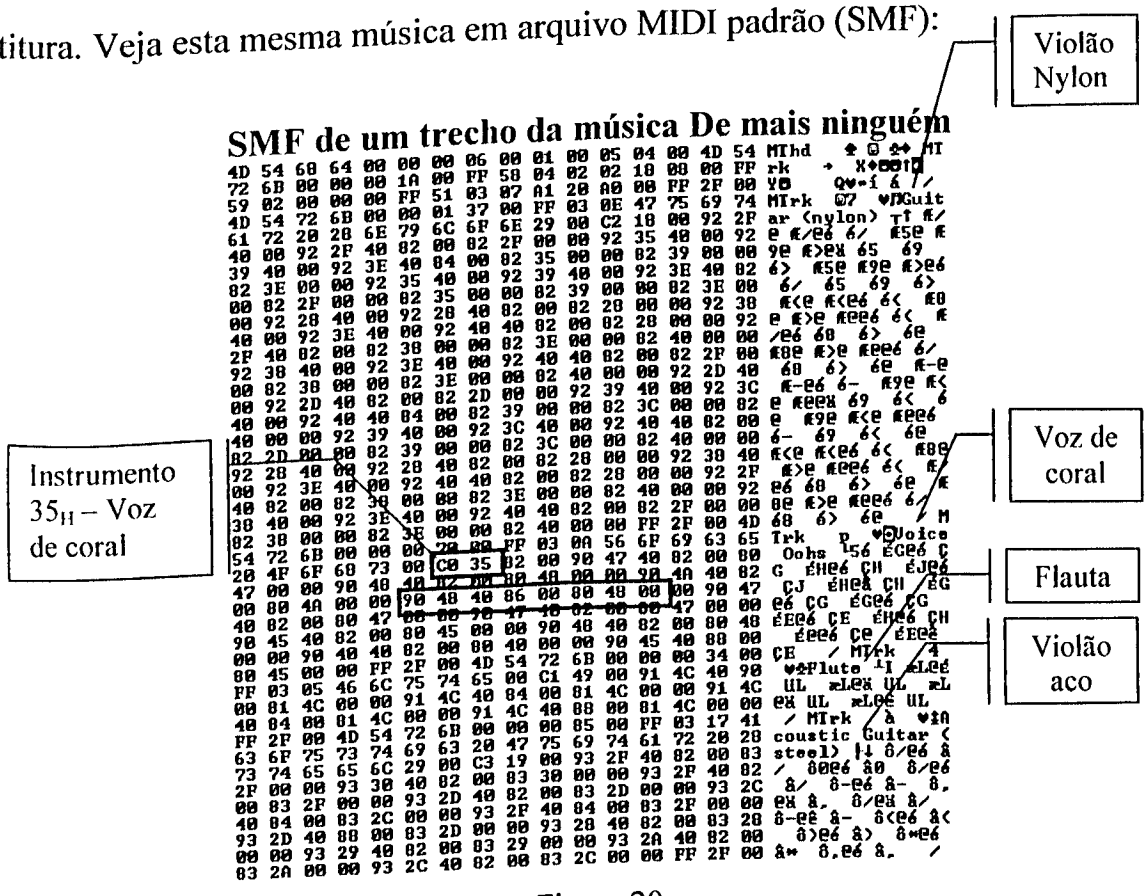


Figura 20

Aparentemente, pode até parecer ser muito mais difícil interpretar e realizar a análise desejada. Esta tarefa não é difícil, apesar de trabalhosa, já que existe um protocolo padrão bem definido permitindo o registro das informações musicais neste tipo de arquivo na 9ª linha da figura 20, a seqüência de código: 90 48 40 86 00 80 48 00, significa que se deve ativar (90) a nota dó da sexta oitava (48) com volume igual a 72 (40_H) e que a mesma deverá ser desativada (80 48 00) após um determinado tempo (86 00). Na 7ª linha está registrado o instrumento para o canal 1 (C0) que é Voz de Coral (35). De igual forma, vários outros detalhes da partitura e da música estão registrados neste arquivo[16].

2.2. Arquivos no formato WAVE

Um arquivo no formato WAVE possui um cabeçalho o qual fornece algumas informações contidas no mesmo, as principais são:

- taxa (freqüência) de amostragem;
- número de canais;
- quantização – resolução em bits;
- tamanho do arquivo em bytes.

2.2.1 Taxa de amostragem:

A taxa de amostragem é um dos parâmetros que definem a qualidade de uma gravação digital. Quanto maior a taxa de amostragem, maior é a fidelidade com que o sinal gravado terá em relação ao som original. Esta taxa de amostragem, para atender os limites da audição humana não necessita exceder 44.100 Hz, de acordo com os critérios definidos por Nyquist, já que os limites das freqüências audíveis estão entre 20Hz e 22kHz[17]. Por este motivo, para a gravação da voz humana a taxa de amostragem necessária é de apenas 22.000 Hz, já que o extensão normal do homem vai 30 Hz a 10.000 Hz.

Comercialmente, as principais taxas de amostragem são as seguintes:

- 11.025 Hz: padrão geralmente utilizado para qualidade de telefone;

---

[16] Ver capítulo sobre MIDI

[17] Este é o motivo da taxa de amostragem de gravação dos CDs musicais ser de 44.100 Hz.

- 22.050 Hz: padrão geralmente utilizado para qualidade de rádio;
- 44.100 Hz: padrão geralmente utilizado para qualidade de CD.
- 32.000 Hz: proporciona bons resultados para fins auditivos, com quantidade superior ao padrão de rádio e telefone e, inferior ao de CD.

2.2.2. Número de canais

Apesar de se possuir sistemas de reprodução quadrifônicos bem como outras configurações utilizadas em equipamentos sofisticados, os arquivos WAVE permitem, atualmente[18], dois tipos de configurações de canais:

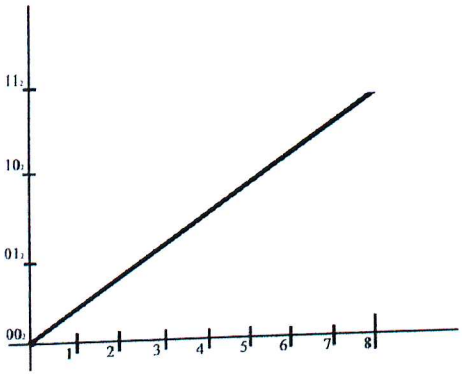
- 1 canal – Monofônico,
- 2 canais – Estereofônico.

2.2.3. Quantização – resolução em bits

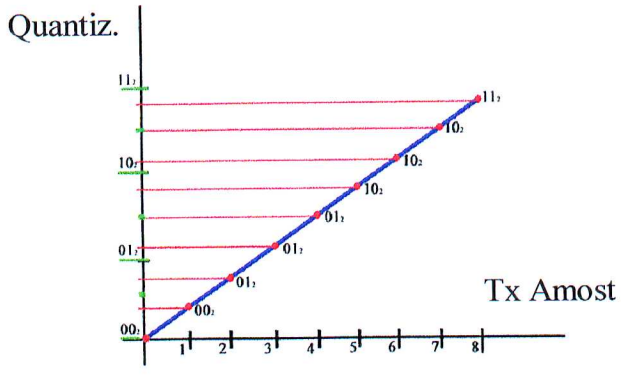
A quantização é um item tão importante quanto a taxa de amostragem para garantir a fidelidade com que o sinal será armazenado no computador e, posteriormente, ser reproduzido. Ela indica quantos bits serão utilizados para representar cada ponto do sinal de áudio que será digitalizado[19] em cada instante da amostragem. A quantização responde a seguinte questão: Como garantir que cada ponto digitalizado reproduza com fidelidade o valor analógico inicial? A seguir, será ilustrado o conceito e os problemas resultantes da quantização.

Dado o sinal analógico da figura 21, adotou-se uma resolução de 2 bits para representá-lo digitalmente e 8 pontos por período do sinal (supondo que o sinal desta figura representa um período do sinal total). A figura 22 mostra a reprodução do sinal digitalizado em sinal analógico.

[18] Já está previsto a expansão destas possibilidades, já que o protocolo possui dois bytes para esta informação.
[19] Digitalizar é o processo de converter um sinal analógico para digital, ou seja, converter um valor em um conjunto de bits que o represente.



Sinal Analógico
Figura 21



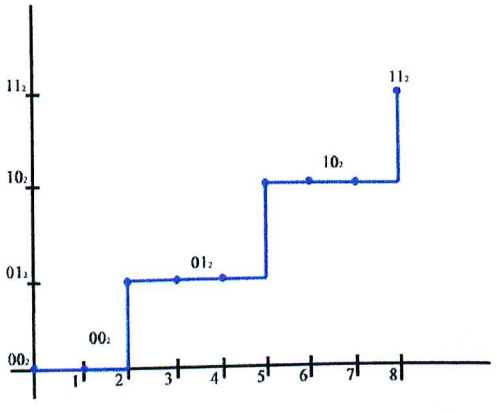
Pontos amostrados
Figura 22

Pontos

0	00
1	00
2	01
3	01
4	01
5	10
6	10
7	10
8	11

Valores Digitalizados

Sinal Digitalizado



Reprodução do sinal digitalizado
Figura 23

O gráfico da figura 22 apresenta uma linha reta inclinada a qual teve seus pontos amostrados digitalizados (quantizados). O ideal seria que cada ponto da amostragem obtivesse um valor distinto ao ser digitalizado, o que não ocorreu[20], para manter a fidelidade do sinal. A figura 22, mostra, assim, a digitalização de 8 amostras a serem quantizadas por uma resolução de dois bits (4 valores possíveis). Estes valores ao serem armazenados pelo computador, serão utilizados para a reconstituição do sinal posteriormente (conversão digital-analógica [12]), a qual é mostrada na figura 23. Pode-se perceber que não há nem semelhança visual entre o sinal original e o reconstituído. Isto se deu devido ter-se registrado os pontos com uma baixa resolução (dois bits), e, portanto, existiram poucos valores binários distintos para representar os valores reais amostrados. Neste caso, o aumento da taxa de amostragem pouco contribuirá para uma melhor fidelidade na amostragem do sinal. O aumento da

[20] Como exemplo, os pontos 2, 3 e 4, ao serem quantizados, obtiveram o mesmo valor.

resolução melhoraria significativamente o sinal digitalizado. A partir de uma determinada resolução (definida pela taxa de Nyquist), ter-se-ia, também, de aumentar a taxa de amostragem. Portanto, a quantização sozinha não garante a qualidade da digitalização do sinal. A finalidade da reprodução é dada pelo binômio taxa de amostragem e quantização.

2.2.4. Tamanho do arquivo

Este item indica quantos bytes de dados possui o arquivo, incluindo os dados do cabeçalho.

2.3. Cabeçalho do arquivo WAVE

As placas de som e os programas que trabalharão os sinais digitalizados precisam conhecer algumas características do protocolo WAVE antes de editá-los ou executa-los, além dos conceitos já apresentados.

- nos arquivos mono e 8 bits por amostra (um Byte) - cada amostra digitalizada ocupa um byte, sendo que os mesmos são armazenados nos arquivos seqüencialmente, ou seja, um após o outro;
- nos arquivos mono e 16 bits por amostra (dois Bytes), o byte mais significativo estará em primeiro lugar e o byte menos significativo estará em segundo lugar;
- nos arquivos estéreos e 8 bits por amostra, o primeiro byte refere-se ao canal direito e o segundo byte refere-se ao canal esquerdo e assim sucessivamente;
- nos arquivos estéreos e 16 bits por amostra, os dois primeiros bytes referem-se ao canal direito e os próximos dois bytes referem-se ao canal esquerdo[21].

2.4. O Arquivo WAVE

Wave	**::**	**Cabeçalho**	**Dados**

[21] Para cada canal o primeiro byte refere-se ao byte mais significativo e segundo byte refere-se ao byte menos significativo.

Cabeçalho	::	RIFF	Tamanho	WAVEfmt	Estrutura	Canais	TxAmostragem
		Tmedia	Qminima	Nbits	Mark	SizeData	

RIFF	::	4 BYTES (0 - 3). Código Hexadecimal da string "RIFF" (52 49 46 46)
Tamanho	::	4 BYTES (4 - 7). Tamanho total do arquivo
WAVEfmt	::	4 BYTES (8 - 15). Código Hexadecimal da string "WAVEfmt" (57 41 56 45 66 6D 74 20)
Estrutura	::	2 BYTES (20-21) - Tipo de Estrutura. PCM (Pulse Code Modulation - Modulação de Código de Pulso)
Canais	::	2 Bytes (22 – 23). Quantidade de Canais. 01: Monofônico 02: Estéreofônico
TxAmostragem	::	4 Bytes (24 – 27). Taxa de Amostragem para PCM: 5000Hz, 11025Hz, 22050Hz, 44100Hz. Comercialmente as principais taxas de amostragem são as seguintes: ♦ 11.025 Hz: padrão geralmente utilizado para qualidade de telefone; ♦ 22.050 Hz: padrão geralmente utilizado para qualidade de rádio; ♦ 44.100 Hz: padrão geralmente utilizado para qualidade de CD. ♦ 32.000 Hz: proporciona bons resultados para fins auditivos, com quantidade superior ao padrão de rádio e telefone e, inferior ao de CD.
Tmedia	::	4 Bytes (28 – 31). Taxa Média de Transferência.
Qminima	::	2 Bytes (32 – 33). Quantidade mínima de bytes utilizados para representar uma amostra. Para PCM: é o número de bytes utilizados para representar uma amostra simples, incluindo os dados para ambos canais, caso seja no formato estéreo. 8 bits mono: 01,

		8 bits estéreo: 02. 16 bits mono: 02, 16 bits estéreo: 04.
Nbits	::	2 Bytes (34 – 35). Número de bits por amostra. 08: oito bists 16: dezesseis bits
Mark	::	4 Bytes (36 – 39). Código Hexadecimal da string "data" (64 61 74 61).
SizeData	::	4 Bytes (40 – 43). Número de bytes de dados a serem lidos.

Dados	::	Bytes

Tabela 1

Capítulo 3

3.1. O Nascimento do MIDI

O MIDI nasceu da necessidade de interligar vários sintetizadores[22] de forma que todos pudessem ser controlados por apenas um músico [52].

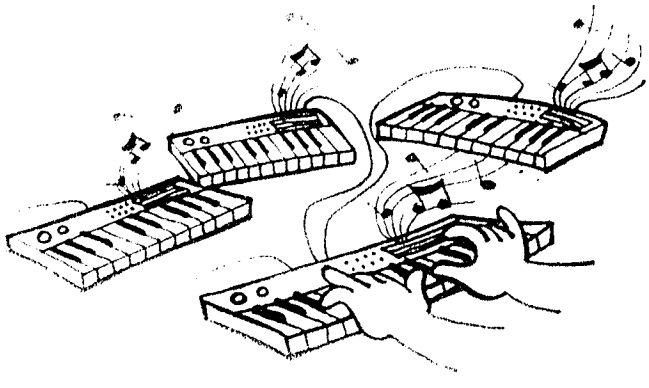


Figura 24

Isto, a princípio, pode parecer estranho nos dias de hoje, já que os sintetizadores atuais simulam orquestras inteiras executadas por um único tecladista ou sequencer[23]. O mesmo não ocorria na década de 70. Tais sintetizadores normalmente eram monotimbrais[24] e monofônicos[25]. Assim, para que um mesmo tecladista pudesse tocar sozinho vários sintetizadores, cada um reproduzindo o som de um instrumento, gerou-se a necessidade de interligá-los.

Esta tarefa não seria em si muito difícil de ser concretizada se o instrumentista utilizasse sintetizadores da mesma marca. Bastaria o fabricante disponibilizar nos

---

22 Instrumento capaz de reproduzir e criar timbres de instrumentos musicais, inclusive de voz humana.

23 Sequencer - equipamento que sincroniza e reproduz vários timbres de instrumentos musicais ao mesmo tempo. Muito utilizado para reproduzir playbacks de músicas MIDI.

24 Monotimbral - sintetizadores que conseguem reproduzir apenas um instrumento (um timbre) de cada vez.

sintetizadores um terminal para interconexão com outros sintetizadores. Alguns fabricantes chegaram até a implementar esta idéia. O que ocorria, e ainda ocorre, é que um músico nem sempre se satisfaz com os timbres gerados por um fabricante apenas. Nestes casos, eles compram várias marcas diferentes que, como tais, possuem circuitos eletrônicos que exigem especialistas em eletrônica para interconectá-los.

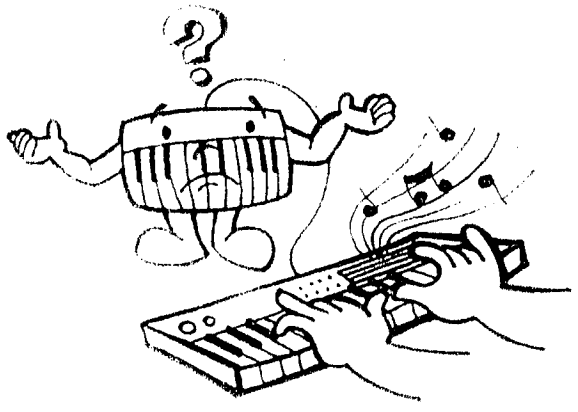


Figura 25

Para resolver este problema de comunicação, no início dos anos 80, três grandes fabricantes - Sequential Circuits, Roland Corporation e Oberheim Electronics - se encontraram na mostra de junho de 1981, da National Association of Music Merchants (**NAMM**) [52] onde decidiram criar uma interface de comunicação padrão que fosse seguida por todos os fabricantes de equipamentos musicais, de forma que qualquer sintetizador ou equipamento futuramente desenvolvido pudesse se comunicar com outro, sem que fosse necessária qualquer modificação.

Neste momento surgiu a idéia do padrão de comunicação MIDI e da interface **USI** (Universal Synthesizer Interface) os quais foram apresentados ao público na mostra do **NAMM** de junho 1982. Em janeiro de 1983, a NAMM publicou a MIDI **Specification 1.0** (Especificação MIDI 1.0) [43] contendo todos os detalhes do protocolo[26] MIDI e da interface de comunicação. Esta especificação foi tão bem elaborada que até hoje é válida e utilizada por todos onde poucos acréscimos foram feitos ao padrão desde sua criação.

---

[25] Monofônico - sintetizador que consegue reproduzir apenas uma freqüência (nota musical) de cada vez.

3.2. MIDI e Computadores

Com a evolução dos computadores pessoais - Mac, Amiga, PCs e outros, o desejo e a necessidade de utilizar o computador como sequencer e como assistente de editoração de partituras surgiram como evolução natural do protocolo MIDI.

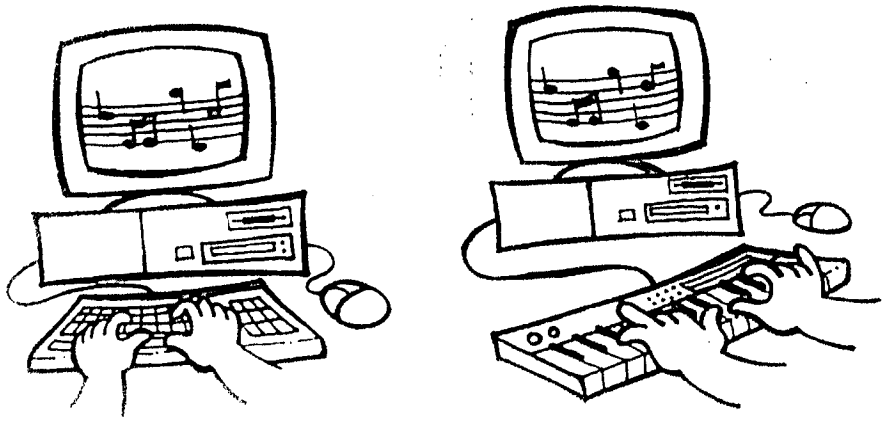


Figura 26

Tendo em vista tais potencialidades, em 1984 a Roland Corporation lançou no mercado uma placa de interface MIDI para computadores, denominada MPU401 [53]. Inicialmente, alguns fabricantes passaram a utilizar essa interface, tais como: Amiga e Mac. Só por volta de 1987, os PCs da IBM tiveram o privilégio de conhecê-la e utilizá-la nos processos de automação musical. Com a utilização de computadores neste processo de interligação de equipamentos por meio de MIDI, houve um crescimento muito grande na qualidade e nas opções que os músicos e gravadoras passaram a usufruir a partir deste momento. Existem atualmente no mercado centenas de marcas e modelos de placas de som equivalentes à MPU401. A *Sound Blaster* é bastante conhecida e possui vários modelos, variando sua polifonia[27] em: 16, 32, 64, 128 e 256 vozes.

---

[26] Protocolo - linguagem de comunicação. Uma ferramenta que permite duas pessoas ou dois dispositivos se comunicarem (Ex. inglês, português, música....MIDI).

[27] Polifonia - capacidade de um sintetizador reproduzir, ao mesmo tempo, mais de uma nota musical. O número de vozes de um sintetizador (módulo de som timbral) define a polifonia.

3.3. O Que é MIDI

MIDI significa: **M**usical **I**nstrument **D**igital **I**nterface, ou seja, Interface Digital para Instrumentos Musicais, isto é, um conjunto de especificações padrão utilizado por fabricantes de instrumentos eletrônicos musicais, ou não, o qual permite que instrumentos de fabricantes diferentes possam ser interligados com total compatibilidade.

- **Interface MIDI**: equipamento ou placa de computador que permite dois sistemas ou equipamentos diferentes se comunicarem de conectores padrões.
- **Mensagem MIDI**: são mensagens enviadas entre equipamentos MIDI por meio das interfaces adequadas. Essas mensagens podem ser de notas musicais ou comandos específicos de configuração de cada equipamento. As mensagens (eventos) musicais mais simples que podem ser transmitidas ou recebidas pelas interfaces MIDI são: ativação e desativação de notas.
- **Dispositivo MIDI**: é todo dispositivo capaz de receber, enviar e interpretar o padrão MIDI. Os mais comuns, encontrados atualmente, são os sintetizadores, baterias eletrônicas, módulos de som e computadores dotados de interface MIDI.
- **Cabos MIDI**: um cabo MIDI é composto de três fios, sendo um para enviar mensagens, outro para receber e outro (uma malha condutora) utilizado como referência para interligação dos circuitos eletrônicos entre dois equipamentos MIDI.
- **Conectores MIDI**: são conectores tipo DIN de 5 pinos, comumentemente utilizados em equipamentos de áudio. Para ligação no cabo MIDI são utilizados apenas 3 destes pinos (os pinos 2, 4 e 5). Os pinos 4 e 5 são conectados ao par de fios e o pino 2 é ligado à malha condutora. Para evitar interferências, é aconselhável utilizar cabos com comprimento máximo de 1,6m. Bons cabos MIDI podem chegar até a 5m. Acima desta metragem não se garante que a informação enviada será recebida corretamente.

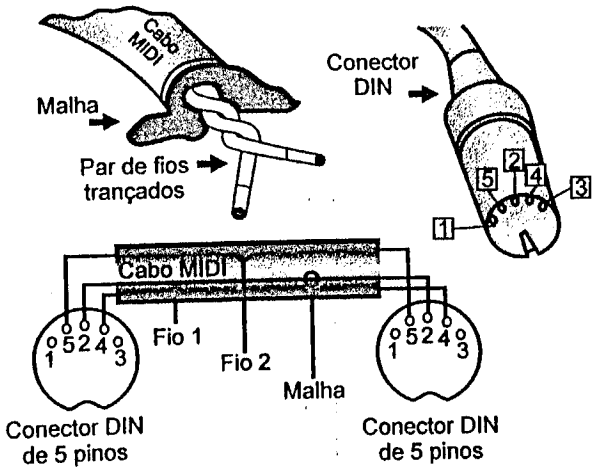


Figura 27

3.4. MIDI Ports

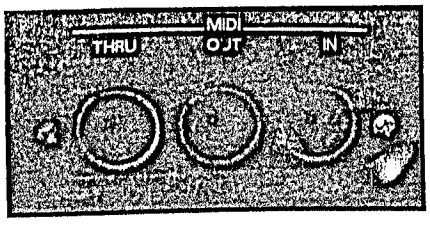


Figura 28

Existem três lugares possíveis para conectar os cabos MIDI em um equipamento MIDI. Esses lugares são denominados MIDI PORT. São eles:

- **MIDI IN**: esse port recebe a informação vinda de outro dispositivo MIDI. Ele apenas recebe informações, não sendo capaz de enviá-las.
- **MIDI OUT**: esse port se encarrega de enviar mensagens de um dispositivo MIDI para outro. Ele só envia mensagens e não é capaz de recebê-las.
- **MIDI TRHU**: esse port retransmite a mensagem recebida no port MIDI IN, ou seja, funciona como uma extensão, uma cópia, da entrada MIDI IN. Assim, MIDI THRU é um port de saída.

Não se deve confundir o port MIDI THRU com o port MIDI OUT. No MIDI OUT são enviadas as mensagens do seu equipamento MIDI, no MIDI THRU são enviadas as mensagens recebidas no MIDI IN. As informações caminham por um cabo MIDI em apenas uma direção. Um único port MIDI pode apenas receber ou transmitir informações, nunca ambas as funções.

3.5. Conexão entre Dispositivos MIDI Utilizando os Ports MIDI IN, MIDI OUT e MIDI THRU

Com exceção dos computadores, todos os dispositivos MIDI são interligados por meio dos cabos MIDI padrões já apresentados anteriormente. A ligação é simples, basta conectar o MIDI IN de um equipamento no MIDI OUT do outro. Ao fazer isto, o equipamento que tiver seu port MIDI OUT ligado no MIDI IN do outro passa a controlá-lo também. Veja a ligação:

O dispositivo MIDI 1 controla o dispositivo MIDI 2

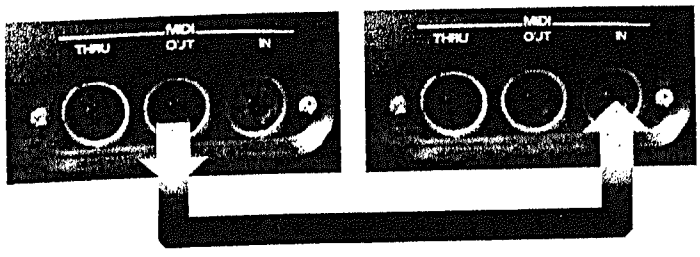


Figura 29

Um mesmo dispositivo pode, também, controlar vários dispositivos. É nesta hora que entra em ação o port MIDI THRU. Devido a atrasos de propagação das mensagens recebidas pelo MIDI IN e repassadas para o MIDI THRU, existe um limite de ligação de no máximo seis dispositivos que podem ser controlados em cascata por um mesmo dispositivo MIDI. Veja essa ligação esquematizada:

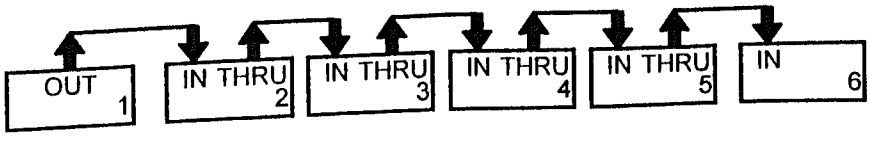


Figura 30

O dispositivo 1 envia a mensagem para o dispositivo 2 por meio do port MIDI OUT que a recebe no port MIDI IN, retransmitindo-a para o dispositivo 3 por meio do port MIDI THRU. O dispositivo 3 repete a ação do dispositivo 2 e assim sucessivamente até que a mensagem seja recebida pelo port MIDI IN do dispositivo 6. Desta forma, cada dispositivo pode executar sua ação sem controlar nenhum dos outros, já que o port MIDI OUT dos dispositivos 2, 3, 4, 5 e 6 não estão ligados a

nenhum outro equipamento. Apenas o dispositivo 1 controlará todos os demais e a si mesmo.

O computador, como já foi dito anteriormente, não pode ser conectado a outros dispositivos MIDI diretamente com um cabo MIDI. Assim, deve-se adquirir uma placa de interface MIDI (por exemplo, uma Sound Blaster) a qual possui um cabo MIDI próprio com um conector extra denominado DB15 (conector do tipo DB com 15 pinos) normalmente utilizado para conexão de joysticks para jogos de computador.

Figura 31

Assim, um cabo MIDI dessas interfaces possui, normalmente, um conector DB15 fêmea para ligar o cabo MIDI ao conector DB15 macho da placa, um conector DB15 macho para ligar no joystick e dois conectores DIN5 para os ports MIDI IN e MIDI OUT (alguns possuem, também, mais um cabo com conector DIN5 para o port MIDI THRU).

Conectores MIDI para Computador

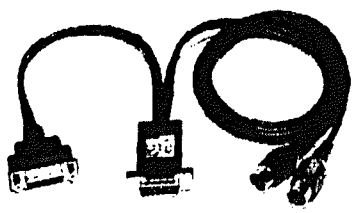


Figura 32

Para interligar um dispositivo MIDI - como, por exemplo, um teclado - a uma placa de som de um computador, é necessário um cabo MIDI com conectores DIN5 em uma extremidade e um conector DB-15 na outra. Para se controlar um teclado via computador, conecta-se, finalmente, o port MIDI OUT do computador ao port MIDI IN do teclado. A conexão do teclado MIDI com o computador fica da seguinte maneira:

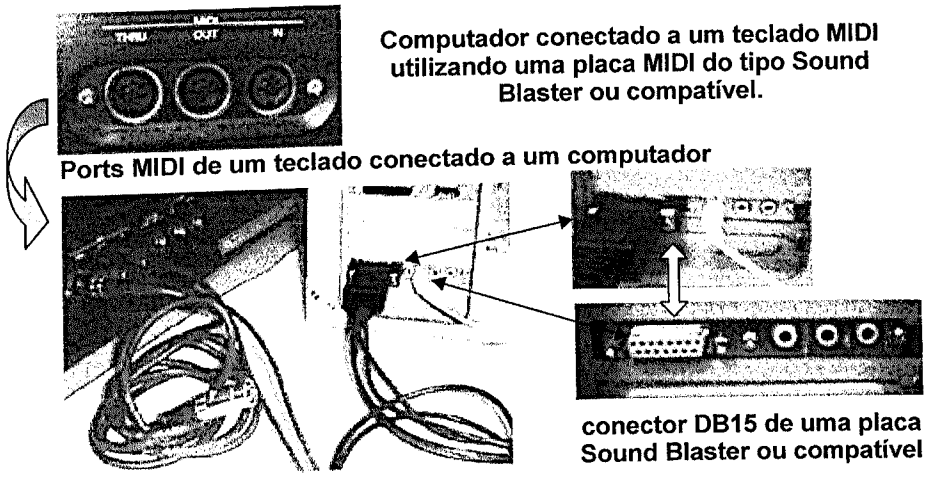


Figura 33

O exemplo a seguir ilustra uma figura esquemática contendo a interligação de um computador e outros cinco dispositivos MIDI quaisquer:

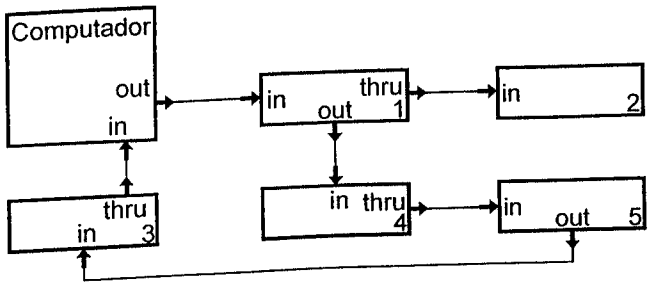


Figura 34

A princípio, esta ligação pode parecer um tanto estranha, no entanto objetiva-se esclarecer como interligar vários dispositivos, utilizando os ports MIDI IN, MIDI OUT e MIDI THRU, e observar o que ocorre quando se faz isto. Para tornar esta tarefa um pouco mais fácil e agradável, vamos analisar em separado cada dispositivo e identificar qual ele controla e por qual ele é controlado.

Qual Controla Qual?

Para um dispositivo controlar outro, ele deve ter seu port MIDI OUT conectado ao port MIDI IN de um equipamento, ou ter seu MIDI OUT retransmitido por um port MIDI THRU.

a). O computador

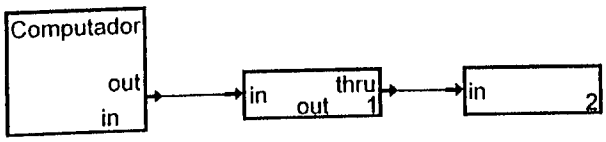


Figura 35

Como podemos observar na figura, o computador tem o seu port MIDI OUT ligado ao port MIDI IN do dispositivo 1 - portanto, ele controla o dispositivo 1. Por sua vez, o port MIDI THRU do dispositivo 1 está ligado ao port MIDI IN do dispositivo 2. Como o MIDI THRU possui uma cópia fiel da entrada MIDI IN do dispositivo 1 que está conectado ao port MIDI OUT do computador, isto indica que o computador também controla, além de si mesmo, o dispositivo 2.

b). O dispositivo 1

Como podemos observar, o MIDI OUT do dispositivo 1 está conectado ao MIDI IN do dispositivo 4 e é retransmitido pelo MIDI THRU do dispositivo 4 para o MIDI IN do dispositivo 5. Assim, o dispositivo 1 está controlando, além de si mesmo, o dispositivo 4 e o dispositivo 5.

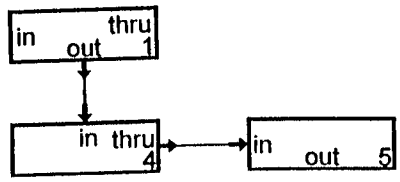


Figura 36

c). O dispositivo 2

Como podemos observar, o port MIDI OUT do dispositivo 2 não está conectado a nenhum outro dispositivo. Portanto, ele não controla ninguém além de si mesmo.

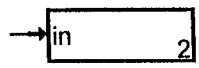


Figura 37

d). O dispositivo 3

Pelo mesmo motivo mostrado no dispositivo 2, o dispositivo 3 também não controla ninguém além dele mesmo.

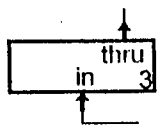


Figura 38

e). O dispositivo 4

Pelo mesmo motivo anterior, o dispositivo 4 não controla ninguém além de si mesmo.

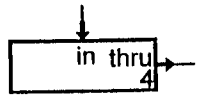


Figura 39

f). O dispositivo 5

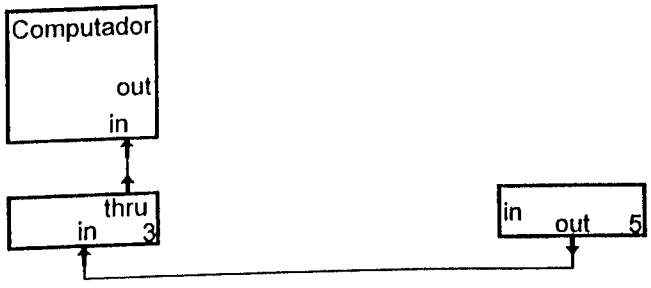


Figura 40

O dispositivo 5 tem o seu port MIDI OUT ligado no MIDI IN do dispositivo 3, e possui seu comando retransmitido ao port MIDI IN do computador por meio de seu port MIDI THRU. Isto indica que o dispositivo 5 controla, além de si mesmo, o dispositivo 3 e o computador.

Podemos, portanto, resumir o que foi analisado na seguinte tabela:

DISPOSITIVO	CONTROLA	É CONTROLADO POR
computador	dispositivo 1 e dispositivo 2	dispositivo 3
dispositivo 1	dispositivo 4 e dispositivo 5	computador
dispositivo 2	ninguém	computador
dispositivo 3	ninguém	dispositivo 5
dispositivo 4	ninguém	dispositivo 1
dispositivo 5	dispositivo 3 e o computador	

Tabela 2

Concluímos que, com um pouco de prática, interligar equipamentos MIDI por meio de seus ports não é uma tarefa tão difícil quando à primeira vista pode parecer.

3.6. O Que Pode ser Feito com MIDI?

A utilização mais comum do padrão MIDI é a sincronização das notas produzidas pelos sintetizadores na reprodução de uma música. Este processo é denominado de seqüenciamento. Por meio de um teclado controlador ou de um computador, pode-se selecionar em cada instrumento o som desejado e a cada instante da execução da música em questão, por meio desse teclado, selecionar o instrumento adequado a cada trecho musical, sem que seja necessário estar trocando de lugar e teclados a todo o tempo.

Além do processo de seqüenciamento de músicas, MIDI facilita os processos de gravação e edição de músicas. Neste processo, como, por exemplo, a edição de uma partitura musical, MIDI envia o código das notas tocadas durante uma execução, as quais são armazenadas conjuntamente com suas características de volume, duração e freqüência.

Caso alguma nota esteja errada devido a uma execução errônea, basta alterar o código da tecla apertada ou algum outro parâmetro e tudo bem. Assim, pode alterar o tempo em que um determinado ritmo foi executado e transpor as tonalidades das músicas com extrema facilidade. Estas tarefas são bastante árduas, quando não impossíveis de ser realizadas em sistemas convencionais.

O padrão MIDI permitiu que se transformasse o computador em estúdios de gravação caseiro de excelente qualidade, trazendo uma economia sem par para os profissionais da área, bem como tornando popular essa arte restrita a poucos antes de sua existência. Assim, os computadores são transformados em gravadores multicanais, seqüenciadores e editores de partituras de excelente qualidade e de fácil manipulação, sem que, com isto, percam sua finalidade para a qual foram anteriormente destinados.

3.7. Como são Gravadas as Músicas MIDI nos Teclados, Seqüenciadores e Computadores?

Para isto um novo padrão teve que ser criado, ou seja, uma nova linguagem, um protocolo, foi criado para que todos os dispositivos MIDI pudessem entender os dados musicais gravados por eles.

Se cada fabricante utilizasse uma forma particular para armazenar as músicas, apenas eles conseguiriam lê-las. Seria como se um japonês escrevesse uma carta para um brasileiro que não entendesse japonês. Neste caso ele não conseguiria entender o que o primeiro estaria querendo lhe dizer. Assim surgiram os padrões, os formatos **SMF** (Standard MIDI Files).

Inicialmente, os SMF formato 0 foram utilizados pelos teclados e seqüenciadores e, posteriormente, com a criação dos SMF formato 1, além dos teclados e sequenciadores eles foram utilizados também pelos editores de notação musical nos computadores. Qualquer um desses formatos permite a reprodução fiel de uma música gravada. Ou seja, toca todos os dezesseis canais MIDI[28], independente do número de tracks[29] que possua.

No caso do MIDI formato 0, ao salvar uma música com 256 tracks, por exemplo, ele grava apenas dezesseis, agrupando todos os tracks que possuírem o mesmo instrumento (mesmo canal MIDI) em apenas um. Isto acontece porque o formato 0 possui apenas um track de gravação. O formato 0 foi criado para os seqüenciadores e teclados poderem reproduzir facilmente e simultaneamente todos os instrumentos utilizados. Daí a explicação de se ter apenas um track. O MIDI formato 1 possui mais de um track, assim, o programa grava a música em tantos tracks quantos forem criados.

3.8. Arquivos MIDI Standard MIDI File (SMF)

Os arquivos SMF vêm ganhando cada vez mais espaço em aplicativos multimídia, principalmente naqueles em que se exige um alto grau de portabilidade e compactação de dados. Assim, com alguns poucos Kbytes de memória pode-se armazenar vários minutos de música que poderão ser reproduzidas por qualquer equipamento MIDI, independendo do fabricante, já que SMF é um padrão internacionalmente aceito.

---

[28] Canais MIDI - cada módulo de som MIDI possui apenas dezesseis canais, ou seja, pode reproduzir no máximo 16 timbres (instrumentos musicais) ao mesmo tempo. Se um equipamento MIDI possuir mais de dezesseis canais, isto indica que ele possui mais de um módulo de som MIDI internamente.

[29] Tracks - pistas de gravação. Um programa pode utilizar quantos tracks desejar, podendo repetir um mesmo instrumento musical em mais de um track. Isto é muito utilizado no seqüenciamento de uma bateria em que cada instrumento dela é gravado em um track à parte.

Por outro lado, muitas vezes um profissional de música deseja implementar programas dedicados a automatizar alguma tarefa que não possa ser realizada por um software já existente, como por exemplo, a implementação de um programa de análise ou composição musical automática baseada em parâmetros e restrições impostas pelo usuário. Como não existe esse programa, alguém deverá criá-lo, e, para tanto, deverá conhecer a estrutura dos SMFs.

Para que se possa entender como é essa estrutura, deve-se, primeiramente, entender o que fazem os SMFs. Eles registram as ações executadas por um intérprete qualquer. Assim, quando um instrumentista pressiona uma tecla em um teclado ou controlador MIDI, o módulo de som associado a ele deve executar esta ação, tocando a nota correspondente enquanto o instrumentista mantiver essa nota pressionada.

O arquivo MIDI (SMF) deve produzir o mesmo efeito quando for lido por um sequenciador. Ele deve simular as mesmas ações efetuadas pelo instrumentista, enviando os eventos ao módulo de som ou outro equipamento MIDI associado, como se eles estivessem ocorrendo naquele momento. Desta forma, os arquivos MIDI SMF nada mais são do que registros digitais dos eventos executados por um músico em um controlador MIDI qualquer. Sendo assim, quando o arquivo MIDI for reproduzido, ele deve mandar seqüencialmente os eventos registrados, indicando quando esses eventos devem ocorrer e quanto tempo devem durar.

3.8.1. Entendendo como o MIDI funciona

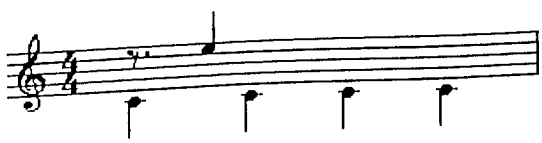


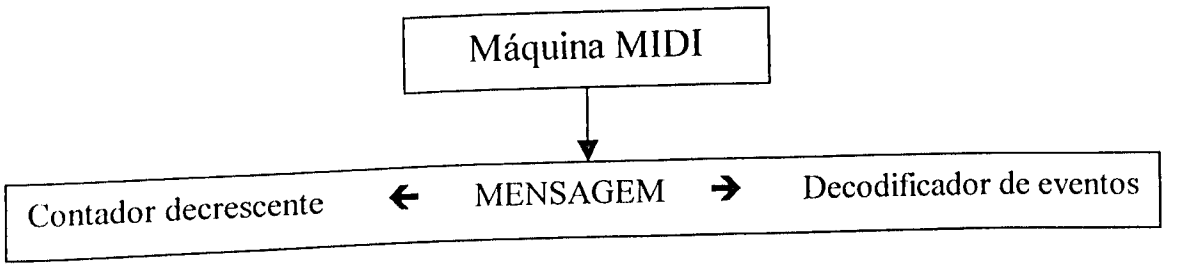
Figura 41

Ao escrever uma partitura, o músico já sabe previamente qual nota deseja grafar e qual sua duração. Observando a partitura, nota-se que não existe um símbolo que indique com precisão qual é o volume, a intensidade sonora com que essa nota deve ser executada. Pode-se escrever embaixo da nota *fortíssimo*, *mezzo forte*, *piano*, *mezzo*

piano, mas estas descrições são subjetivas, ou seja, cada intérprete executará um *fortíssimo* ou qualquer outro sinal de dinâmica diferente de outro. Um ponto forte do padrão MIDI é que ele registra com precisão o tempo de duração da nota e não sua figura musical, bem como volume executado pelo intérprete.

3.8.2 Arquitetura da máquina MIDI

A máquina possui uma unidade de aquisição de um byte de mensagem, um contador decrescente e um decodificador de eventos.



O processamento de uma mensagem segue o seguinte protocolo:

1. A máquina recebe um ou mais bytes de mensagem com a contagem a ser armazenada no contador decrescente – Contador de *Delta-Times*;
2. Ela recebe a mensagem contendo o evento a ser decodificado e executado;
3. Quando o contador chega em zero, o evento é disparado;
4. Repetem-se os itens 1,2 e 3 até que seja encontrada uma mensagem de fim de arquivo (3 bytes = FF2F00).

Para se trabalhar com arquivos MIDI SMF, deve-se manipular corretamente os tempos do contador decrescente. Estes tempos são denominados na máquina MIDI por *Delta-Times*. Eles são armazenados internamente em contadores de 7 bits, e, sendo assim, cada módulo deste contador pode armazenar uma contagem de 0_{10} a 127_{10} (00_{H} a $7F_{H}$). Quando for necessário utilizar valores de contagens maiores que 127_{10}, dever-se-á utilizar mais de um módulo do contador para armazená-lo. O número máximo de módulos do contador decrescente é 4 (quatro). Uma mensagem MIDI possui 8 bits (um byte), sendo assim, quanto um *Delta-Time* é enviado, o oitavo bit (o mais significativo) é que indica à máquina MIDI se o mesmo possui um ou mais bytes.

Portanto, se o bit mais significativo da mensagem for 1 (um), isto indica que o *Delta-Time* possuirá mais de um byte. Se for 0 (zero), indicará que o *Delta-Time* possuirá apenas um byte. Se o bit mais significativo do primeiro byte do *Delta-Time* for 1, significa que o mesmo terá pelo menos mais um byte. Se o bit mais significativo do próximo byte for 0 (zero) isto significa que este é o último byte do presente *Delta-Time*, se for 1 (um), indica que o *Delta-Time* terá mais um byte. O raciocínio se repete até que se obtenha, no máximo, um *Delta-Time* com 4 bytes. A tabela 3 ilustra este enunciado:

Bit mais significativo		
0110 0001:	um byte	*delta-time* com um byte.
1000 1001:	primeiro byte	*delta-time* com quatro bytes.
1101 1100:	segundo byte	
1110 1101:	terceiro byte	
0110 0110:	quarto byte	

Tabela 3

3.8.3. Representando uma nota musical e sua duração em Delta-Times

A máquina MIDI recebe um valor de *Delta-Time* e, logo após, recebe um evento a ser executado quando o contador chegar em zero, desta forma para tocar uma nota musical, a máquina MIDI deverá receber uma mensagem de ativação de nota [30] [39] [40]. Quando a nota for ativada ela tocará indefinidamente até que a máquina MIDI receba uma nova mensagem dizendo que esta nota deve ser desativada. A máquina MIDI permite que você utilize 16 canais diferentes para tocar sua nota musical [30] [39] [40]. Cada canal pode utilizar um instrumento diferente de cada vez. Assim, para gravar uma nota musical em um arquivo SMF, para posterior execução, deve-se proceder da seguinte forma:

1. Enviar uma mensagem de um a quatro bytes, com um *Delta-Time* informando quando a nota deverá começar a tocar. No caso da nota começar instantaneamente, este *Delta-Time* deverá ser 0;

2. Enviar uma mensagem (um byte) indicando qual dos 16 canais deverá ativar (tocar) a nota desejada;
3. Enviar uma mensagem (um byte) com o evento que informe qual nota deverá ser ativada;
4. Enviar uma mensagem (um byte) informando com qual intensidade (volume) esta nota deverá ser ativada;
5. Enviar um novo *Delta-Time* (um a quatro bytes) com o tempo de duração da nota (quando o *Delta-Time* chegar em zero o evento seguinte é ativado);
6. Enviar uma mensagem (um byte) com o evento de desativar nota em um determinado canal;
7. Enviar uma mensagem (um byte) contendo o código da nota a ser desativada;
8. Enviar uma mensagem (um byte) com o volume desta nota (pode ser qualquer valor, já que a nota vai ser desativada).

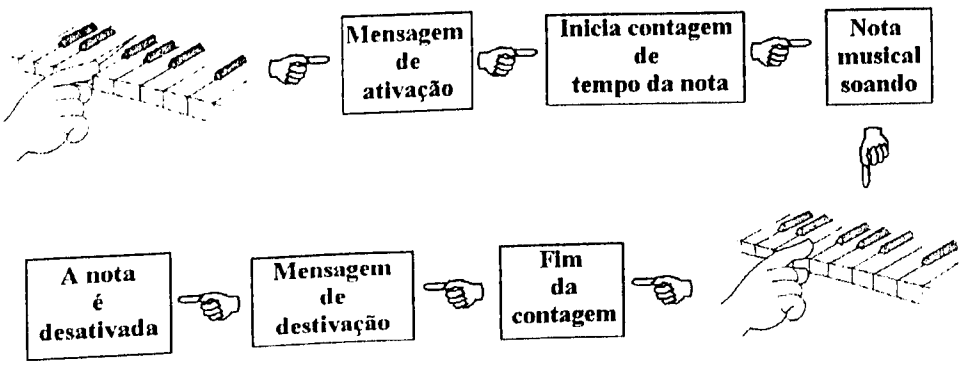


Figura 42

Para armazenar estas durações, foi criado no padrão MIDI uma estrutura chamada de Delta-Time. Os Delta-Times são calculados tomando como base a duração de uma semínima. Assim, a máquina MIDI cria internamente uma unidade de tempo que define quantos pulsos (batidas) de seu relógio equivalerá ao tempo de uma semínima. Pulsos por semínima é a denominação dada pelo padrão MIDI de ppq (pulses per quarter-note). Conhecendo o valor da ppq adotada, pode-se calcular os Delta-Times de cada nota executada.

3.8.4. Pulses Per Quarter-note - ppq

ppq, significa pulses per quarter-note (pulsos por semínima), ou seja, em quantas partes (pulsos) será dividida uma semínima. O valor máximo de uma ppq adotado pela maioria dos programas é 480. A ppq é um indicativo da precisão com que uma nota musical foi ou será gravada em um arquivo MIDI. Por exemplo:

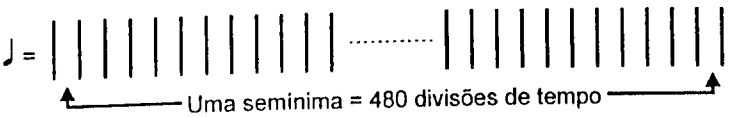


Figura 43

- **Um ppq** = 2, indica que temos uma precisão de captura igual a uma colcheia (duas colcheias = uma semínima → ppq = 2 = 2 colcheias por semínima).

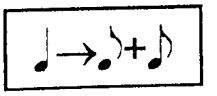


Figura 44

- Assim, para uma precisão de uma semicolcheia, deve-se ter uma ppq = 4, ou seja, 4 semicolcheias por semínima.

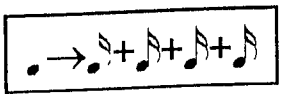


Figura 45

Concluindo:

- Se o ppq = 1 → precisão (captura) de semínimas
- Se o ppq = 2 → precisão (captura) de colcheias
- Se o ppq = 4 → precisão (captura) de semicolcheias
- Se o ppq = 8 → precisão (captura) de fusas
- Se o ppq = 16 → precisão (captura) de semifusas
- Se o ppq = 32 → precisão (captura) de quartifusas

Portanto uma ppq de 480 é uma precisão realmente boa, ou seja, de 1/480 de uma semínima.

3.8.5. Delta Time

O Delta-Time indica quanto tempo o dispositivo, que está lendo e executando o arquivo MIDI, deverá esperar para iniciar a execução do evento que se segue. Delta-time e ppq representam a mesma grandeza. A diferença é que ppq utiliza os oito bits[30] de um byte para representar o tempo de uma nota e o delta-time apenas sete. Assim, faz-se necessário criar uma regra de conversão de ppq para delta-time e vice-versa.

Para entender melhor a conversão de ppq em Delta-Time, em primeiro lugar se faz necessário saber o porquê da criação dos Delta-Times e o motivo de não se utilizar os oito bits do byte (apenas sete). Se o tempo das notas será armazenado pelo computador ou processador dos equipamentos MIDI em forma de bytes. A questão é: Como saber quantos bytes definirão o tempo de uma nota? Será que apenas um byte é o bastante?

Para responder a estas duas perguntas, adotou-se como exemplo uma **ppq** = 480_{10} = $1E0_{11}$[31]. O maior valor que um byte pode representar é quando todos os seus bits forem iguais a 1, ou seja, 255_{10}.

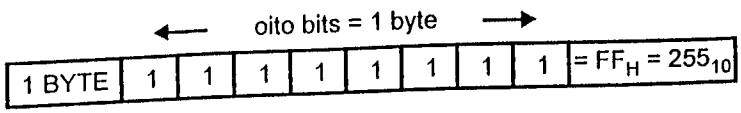


Tabela 4

Desta forma, o maior Delta-Time que se pode representar com um byte seria 255. Com este valor não é possível representar sequer o tempo de uma semínima, para uma ppq de 480.

Como representar valores maiores que o do exemplo e, como o computador ou teclado, pode saber que o Delta-Time será representado com mais de um byte?

---

[30] Bit – pode assumir apenas dois valores: o **0** e o **1**. Um **byte** é um conjunto de 8 bits.
[31] Ver o Anexo "Sistemas de Numeração"

Este problema foi solucionado utilizando o bit mais significativo[32] de cada byte do Delta-Time para informar ao sistema (computador ou teclado) de quantos bytes será formado o presente delta-time. Ao utilizar esse oitavo bit como sinalizador, passa-se a só utilizar sete bits para armazenar a contagem em cada byte. Para uma ppq de 480 isto funciona da seguinte maneira:

Já apenas que um byte não é suficiente para representar ppq maior que o máximo valor de contagem com sete bits (127), se faz necessário, portanto, utilizar dois bytes para armazenar a contagem do ppq. Colocando-se as potências de 2 equivalentes em cada casa binária [24], agrupando os dois bytes, já que eles vão formar um único número, obtem-se o seguinte resultado:

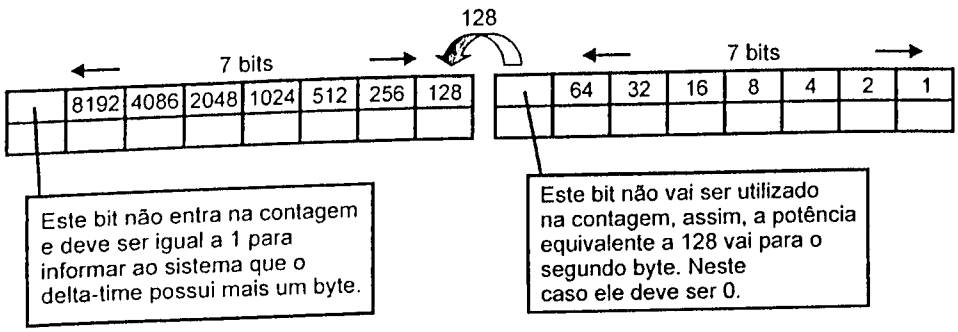


Tabela 5

Assim, 480 convertidos em binário é igual a 256 + 128 + 64 + 32. Deve-se colocar, portanto, bits 1 nas respectivas casas e 0 nas demais, ou seja:

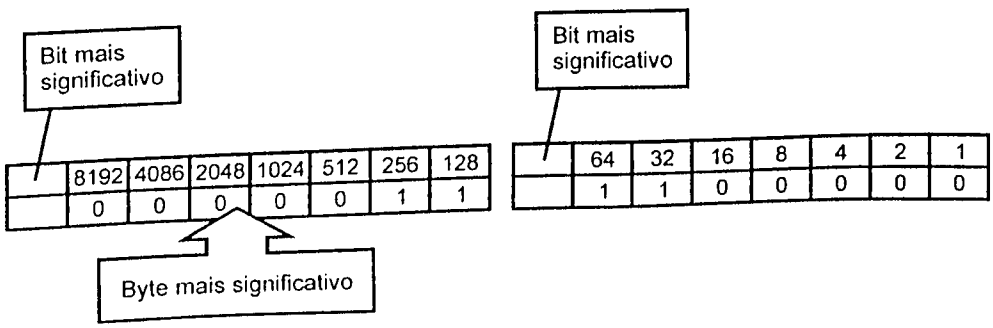


Tabela 6

Como completar os bits da oitava casa , que são os bits mais significativos de cada byte?

[32] O bit mais significativo de um byte é o oitavo, ou seja, o bit mais à esquerda do byte.

No byte mais significativo, deve-se informar que o Delta-Time é formado por mais de um byte, desta forma, coloca-se um bit 1 na oitava casa deste byte. Este bit indica ao sistema (computador ou teclado) que existe mais um byte no Delta-Time.

	8192	4086	2048	1024	512	256	128
1	0	0	0	0	0	1	1

	64	32	16	8	4	2	1
	1	1	0	0	0	0	0

1 se houver mais um Byte e **0** se não houver mais nenhum.

Tabela 7

No segundo byte, se houvesse mais um byte para o delta-time, dever-se-ia também colocar também o bit mais significativo igual a 1. Como neste caso o próximo byte é o último, ou seja, o delta-time não terá mais nenhum byte, este bit deve ser 0.

	8192	4086	2048	1024	512	256	128
1	0	0	0	0	0	1	1

	64	32	16	8	4	2	1
0	1	1	0	0	0	0	0

Tabela 8

Agora, de posse dos dois bytes completos, basta ler o valor final e convertê-lo em hexadecimal. Para tanto, deve-se esquecer agora as potências de 2 e analisar cada byte em separado, convertendo-se cada 4 bits de cada byte em hexadecimal, conforme a tabela seguinte:

Primeiro Byte							
4 bits				4 bits			
8	4	2	1	8	4	2	1
1	0	0	0	0	0	1	1

Segundo Byte							
4 bits				4 bits			
8	4	2	1	8	4	2	1
0	1	1	0	0	0	0	0

8	3
83	

6	0
60	

Tabela 9

Assim, uma ppq de 480 equivale a um Delta Time com dois bytes cujo valor em hexadecimal é igual a 83 60.

- Com este exemplo conclui-se que a maior ppq para um byte é $7F_H = 127_{10}$, o maior valor que podemos grafar com sete bits, já que, neste caso, o oitavo bit deve ser zero. Então, delta-time com um byte, o oitavo bit deve ser zero, o que indica que o delta-time só possui um byte, este byte.

	64	32	16	8	4	2	1	$= 7F_H = 127_{10}$
0	1	1	1	1	1	1	1	

Delta-time com um byte.

Tabela 10

- Pode-se concluir, também, que a maior ppq para dois bytes é 16.383, com um Delta-Time = FF 7F, conforme a tabela seguinte:

	8192	4086	2048	1024	512	256	128
1	1	1	1	1	1	1	1

	64	32	16	8	4	2	1
0	1	1	1	1	1	1	1

Tabela 11

16.383 = 8182 + 4086 + 2048 + 1024 + 512 + 256 + 128 + 64 + 32 +16 + 8 + 4 + 2 + 1.

Esta ppq convertida em delta-time é FF 7F.

- Com três bytes tem-se que o maior valor de ppq é de 2.097.151, com um Delta-Time de FF FF $7F_H$.
- Com quatro bytes, número máximo de bytes que o delta-time pode ter, tem-se uma ppq de 268.435.455 cujo maior valor de delta-time é FF FF FF $7F_H$.

Convertendo ppq em delta-time, para o tempo de uma colcheia, novamente com uma ppq = 480 pulsos, significa que uma colcheia (metade de uma semínima) terá 240 pulsos. Novamente, pode-se constatar que com apenas um byte não é possível representar o valor de uma colcheia (máximo valor de um byte = 127). Como 240 é menor que 16.383 (maior valor de uma ppq com 2 bytes), tem-se então que dois bytes são suficientes para representar tal colcheia. Assim, o delta-time de uma colcheia (240 pulsos) será:

	8192	4086	2048	1024	512	256	128
1	0	0	0	0	0	0	1

	64	32	16	8	4	2	1
0	1	1	1	0	0	0	0

Tabela 12

Ou seja: 240 = 128 + 64 + 32 + 16. Assim, o delta-time fica:

Primeiro Byte							
4 bits				4 bits			
8	4	2	1	8	4	2	1
1	0	0	0	0	0	0	1

Segundo Byte							
4 bits				4 bits			
8	4	2	1	8	4	2	1
0	1	1	1	0	0	0	0

8	1
81	

7	0
70	

Tabela 13

Ou seja, o delta-time de $240_{10} = 81\ 70_{H}$.

3.8.6. Analisando os arquivos MIDI SMF

O problema na geração dos SMF está focado no cálculo dos *Delta-Times*. Estes são calculados tomando como base a duração de uma semínima[33]. Assim, a máquina MIDI deverá conhecer quantos pulsos de relógio (unidade de tempo mínima da máquina) equivalerão ao tempo de uma semínima. Esta base de tempo (pulsos por semínima) é denominada pelo padrão MIDI por ppq (pulses per quarter note). Conhecendo o valor da ppq adotada pode-se calcular os *Delta-Times* equivalentes para cada nota.

3.8.7. Convertendo ppq para Delta Time

Adotando-se uma ppq = 96_{10} (60_{H}), ou seja, a semínima é igual a 60_{H}, as demais figuras musicais possuirão os seguintes valores:

Figura Musical	Nome em Português	Nome em Inglês	ppq
	Breve	Double Whole	768_{10} (300_{H})
	Semibreve	Whole	384_{10} (180_{H})
	Mínima	Half	192_{10} ($C0_{H}$)
	Semínima	**Quarter Note**	**96_{10} (60_{H})**
	Colcheia	Eighth	48_{10} (30_{H})

[33] Semínima = quarter note.

	Semicolcheia	Sixteenth	24_{10} (18_{H})
	Fusa	32nd	12_{10} (C_{H})
	Semifusa	64th	6_{10} (6_{H})
	Quarto de Fusa	128th	3_{10} (3_{H})

Tabela 14

Para calcular o valor do Delta-Time da semibreve (384_{10} (180_{H})), basta dividir a ppq por 128. A parte inteira da divisão indicará a contagem do primeiro byte (mais significativo) e o resto da divisão indicará a contagem do segundo byte (menos significativo).

Exemplo: 384_{10} : 128_{10} = 3_{10} como o resto da divisão foi igual a zero, tem-se que o primeiro byte possui uma contagem igual a 3 e o segundo byte igual a zero. Assim o valor do Delta-Time da semibreve é igual a 8300_{H} (1000 0011 0000 0000).

Quando o resto da divisão por 128 for maior que 127, deve-se gerar mais um byte para compor o respectivo Delta-Time, obedecendo-se o limite máximo de 4 bytes.

3.8.8. A diferença entre Formato 0 e Formato 1

No formato 0 todas as notas de todos os canais são registradas no arquivo SMF em apenas um *track* conforme forem aparecendo na música. No formato 1 cada canal é registrado no arquivo SMF em um *track* independente.

3.9. O Arquivo MIDI SMF Formato 1

SMF	::	**Cabeçalho Principal**		**Cabeçalho dos Tracks**		
Cabeçalho Principal	::	Chunk type	Tamanho	Formato	Ntracks	TipoDeltaTime
Cabeçalho dos Tracks	::	Chunk type	Tamanho	MTrkEventos		

Cabeçalho Principal

Chunk type	Tamanho	Formato	Ntracks	TipoDeltaTime

Chunk type	::	4 bytes: Mthd = 4D 54 68 64.
Tamanho	::	Tamanho do Cabeçalho Principal: 6 bytes, sendo 2 bytes para indicar o Formato, 2 bytes para indicar Número de Tracks, 2 bytes para indicar o Tipo de Delta-Time.
Formato	::	Indica o Formato com 2 bytes.: Formato 0 = 00 00, Formato 1 = 00 01, Formato 2 = 00 10.
Ntracks	::	Número de Tracks da música, sendo um track para cada canal MIDI e mais um para as configurações (metrônomo, armadura de clave, fórmula de compasso, etc). Possui 2 bytes.
TipoDeltaTime	::	Indica o Tipo de Delta Time. Possui 2 bytes. Se o bit mais significativo for 0 (zero) o Delta-Time será do tipo ppq. Se o bit mais significativo for 1 será do tipo SMPTE.

Cabeçalho dos Tracks

Chunk type	Tamanho	MTrkEventos

Chunk type	::	4 bytes: Mtrk = 4D 54 72 6B.

Tamanho	::	Possui 4 bytes. Indica a soma de todos os bytes do track, incluindo os 3 bytes indicativos do fim do track: FF 2F 00.

MtrkEventos	::	DeltaTime	Eventos
Deltatime	::	Possui de 1 a 4 bytes. Indica quanto tempo o dispositivo, que está lendo e executando o arquivo MIDI, deverá esperar para iniciar a execução do evento que o segue.	

Eventos	::	EventosMIDI	EventosSysex	MetaEventos
EventosMIDI	::	A quantidade de bytes dependerá do tamanho do arquivo. Evento MIDI é qualquer mensagem de canal, ou seja, eventos de notas e os controles aplicados à elas.		
EventosSysex	::	$F0_H$	Tamanho	BytesTrans
		São eventos utilizados para mandar mensagens excluisvas para um determinado equipamento.		
Tamanho	::	O número de bytes dependerá do tamanho do Evento Sysex		

BytesTrans	::	São os Bytes Transmitidos pelo Evento Sysex. Deverá terminar com o byte $F7_H$.		
MetaEventos	::	Tipo	Tamanho	Texto
		São eventos não-MIDI contendo informações úteis, e necessárias para os equipamentos que executarão os eventos MIDI. Um Meta-Evento inicia-se com o byte FF. Os equipamentos que não reconhecem todos os tipos de Meta-Eventos devem ignorá-los sem emitir mensagem de erro.		
Tipo	::	A quantidade de bytes dependerá do tipo de Meta-Evento. Os principais são: FF 51 = Set Tempo (Metrônomo). Possui 3 bytes. FF 58 = Fórmula de Compasso (Time Signature). Possui 4 bytes FF 59 = Armadura de Clave (Key signature). Possui 2 bytes. FF 2F = Fim de Track. FF 01 = Texto. A quantidade de bytes varia de acordo com o texto. FF 04 = Nome do Instrumento. A quantidade de bytes varia de acordo com o instrumento. FF 05 = Letra da Música. A quantidade de bytes varia de acordo com o tamanho da letra. FF 06 = Marcas. A quantidade de bytes é variável.		
Texto	::	A quantidade de bytes varia de acordo com o texto.		

Tabela 15

3.10. Meta-Eventos

3.10.1. Meta-Evento Set Tempo:

FF 51 03 Tempo

Metrônomo = Semínimas por minuto

$$\text{Metrônomo} = \frac{60\ (1\ \text{minuto})}{\text{Tempo em Segundos}}$$

Exemplo: FF 51 03 098968

Tempo = 098968_H = 625.000 Microssegundos = 0,625

$$\text{Metrônomo} = \frac{60}{0{,}625} = 96\ \text{semínimas por minuto}$$

3.10.2. Meta-Evento Fórmula de Compasso (Time Signature):

FF 58 Tamanho

Tamanho = 4 bytes:

nn = Numerador.

dd = Potencia de 2 do denominador.

cc = Número de MIDI clocks no metrônomo por semínima (unidade de tempo do compasso, ou seja, semínima = 24_{10} (18_{H}).

bb = Número de fusas por unidade de tempo.

Fórmula de Compasso = nn/dd

Exemplo1: FF 58 04 06 03 0C 08

FF 58 = Meta Evento Fórmula de Compasso,

04_{H} = Tamanho,

06_{H} = nn = 6,

03_{H} = dd = 2^3 = 8,

$0C_{H}$ = cc = 12_{10} (colcheia),

08_{H} = bb = 8 fusas.

Fórmula de Compasso = nn/dd = 6/8.

Exemplo2: FF 58 04 02 02 18 08

FF 58 = Meta Evento Fórmula de Compasso,

04_{H} = Tamanho,

02_{H} = nn = 2,

02_{H} = dd = 2^2 = 4,

18_{H} = cc = 24_{10} (semínima),

08_{H} = bb = 8 fusas.

Fórmula de Compasso = nn/dd = 2/4.

3.10.3. Meta-Evento Armadura de Clave (Key Signature):

FF 59 Tamanho

Tamanho = 2 bytes:

sf = Acidentes da Armadura de Clave.

mi = Modo: Maior ou menor.

Tonalidades com #		Hexa		Tonalidades com #	Hexa	
	Maiores	sf	mi	menores	sf	mi
0	Do Maior	00	00	La menor	00	01
1 #	Sol Maior	01	00	Mi menor	01	01
2 #	Re Maior	02	00	Si menor	02	01
3 #	La Maior	03	00	Fa# menor	03	01
4 #	Mi Maior	04	00	Do# menor	04	01
5 #	Si Maior	05	00	Sol# menor	05	01
6 #	Fa# Maior	06	00	Re# menor	06	01
7 #	Do# Maior	07	00	La# menor	07	01

Tabela 16

Tonalidades com *b*		Hexa		Tonalidades com *b*	Hexa	
	Maiores	sf	mi	menores	sf	mi
1 *b*	Fa Maior	FF	00	Re menor	FF	01
2 *b*	Sib Maior	FE	00	Sol menor	FE	01
3 *b*	Mib Maior	FD	00	Do menor	FD	01
4 *b*	Lab Maior	FC	00	Fa menor	FC	01
5 *b*	Reb Maior	FB	00	Sib menor	FB	01
6 *b*	Solb maior	FA	00	Mib menor	FA	01
7 *b*	Dob Maior	F9	00	Lab menor	F9	01

Tabela 17

3.10.4. Exemplo de arquivo MIDI SMF – Formato 1

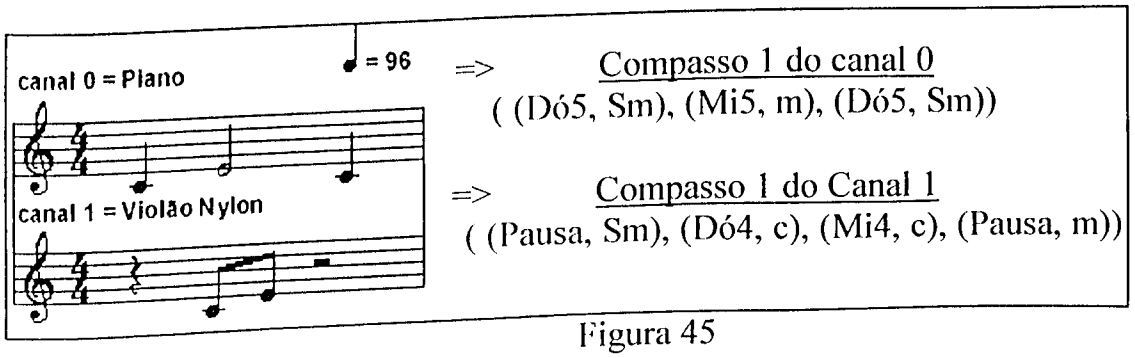


Figura 45

Cabeçalho principal

4D 54 68 64	Mthd
00 00 00 06	seguem 6 bytes do cabeçalho principal
00 01	formato 1
00 04	quatro tracks
00 60	ppq = 60_{H} (96_{10})

Cabeçalho do primeiro track – parâmetros iniciais

4D 54 72 6B	MTrk

00 00 00 19	seguem 25 bytes do presente track (todos os bytes até FF2F00)
00	delta-time = 0
FF 58	Meta Evento – Fórmula de Compasso
04 04 02 18 08	Compasso 4/4
00	Delta-time = 0
FF 59	Meta Evento – Armadura de Clave
02 00 00	Dó maior
00	Delta-time = 0
FF 51	Meta Evento – Set Tempo
03 09 89 68	Metrônomo = 96 batidas por minuto (1 : (625000 : 60)) x 1.000.000. Obs. $098968_{11} = 625000_{10}$
00	Delta-time = 0
FF 2F 00	Fim de track

Cabeçalho do segundo track – canal 0 - piano

4D 54 72 6B	MTrk
00 00 00 1E	seguem 30 bytes do presente track (todos os bytes até FF2F00)
00	Delta-time = 0
C0 00	C = Mudar instrumento; 0 = Canal 0; 00 = Piano
00	Delta-time = 0
90	Ativar nota no canal 0
3C 50	3C =Nota Dó5 ; 50_{11} = volume = 80_{10}
60	Delta-time = duas colcheias = uma semínima
80	Desativar nota no canal 0
3C 00	$3C_{11}$ =Nota Dó5 ; 00_{11} = volume = 00_{10}
00	Delta-time = 0
90	Ativar nota no canal 0
40 50	Ativar nota Mi5 (40) com volume 80_{10}
8140	Delta-time de uma mínima (2 semínimas) = 192. 192 : 128 = 1 inteiro e sobra 64_{10} = 40H-> dois bytes de delta-time com o bit mais significativo do primeiro byte = 1. Primeiro byte = 1000 0001 e segundo byte = 0100 0000 => Delta-time = 81 40
80	Desativar nota no canal 0
40 00	40_{11} = Nota Mi5; 00_{11} = volume = 00_{10}
00	Delta-time = 0
90	Ativar nota no canal 0
3C 50	3C =Nota Dó5 ; 50_{11} = volume = 80_{10}
60	Delta-time = uma semínima
80	Desativar nota no canal 0
3C 00	$3C_{11}$ =Nota Dó5 ; 00_{11} = volume = 00_{10}
00	Delta-time = zero
FF 2F 00	Fim do track

Cabeçalho do terceiro track – canal 1 - violão

4D 54 72 6B	MTrk
00 00 00 16	seguem 22 bytes do track
00	Delta-time = 0
C1 18	C = Mudar instrumento; 1 = Canal 1; 18 = Violão de Nylon
60	Delta-time de uma semínima
91	ativar nota no canal 1
3C 50	Nota Dó4, volume 80_{10}
30	Delta-time de uma colcheia
81	Desativar nota no canal 1
3C 00	$3C_{11}$ =Nota Dó4; 00_{11} = volume = 00_{10}
00	Delta-time = 0
91	Ativar nota no canal 1
40 50	Ativar nota Mi4 (40) com volume 80_{10}
30	Delta-time de uma colcheia
81	Desativar nota no canal 1
40 00	40_{11} = Nota Mi4; 00_{11} = volume = 00_{10}
00	Delta-time = 0
FF 2F 00	Fim do track

3.10.5. Exemplo de arquivo MIDI SMF – Formato 1 – com Runing Status

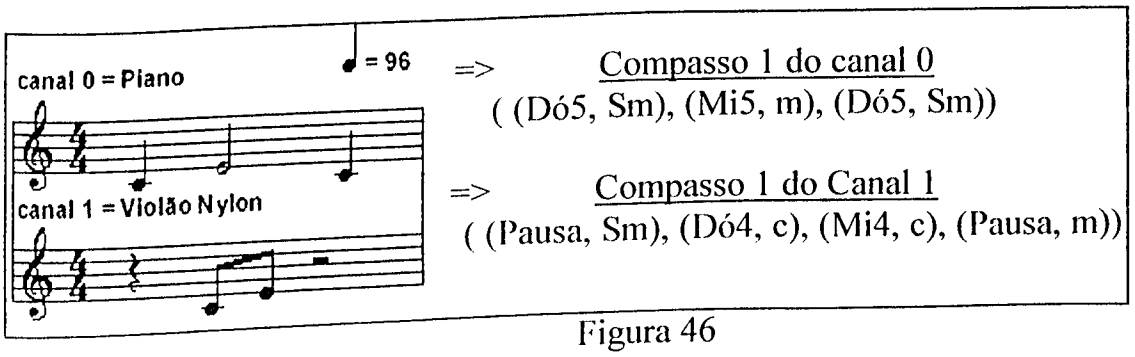


Figura 46

Cabeçalho principal

4D 54 68 64	Mthd
00 00 00 06	seguem 6 bytes do cabeçalho principal
00 01	formato 1
00 04	quatro tracks
00 60	ppq = 60_{11} (96_{10})

Cabeçalho do primeiro track – parâmetros iniciais

4D 54 72 6B	MTrk
00 00 00 19	seguem 25 bytes do presente track (todos os bytes até FF2F00)
00	delta-time = 0
FF 58	Meta Evento – Fórmula de Compasso

04 04 02 18 08	Compasso 4/4
00	Delta-time = 0
FF 59	Meta Evento – Armadura de Clave
02 00 00	Dó maior
00	Delta-time = 0
FF 51	Meta Evento – Set Tempo
03 09 89 68	Metrônomo = 96 batidas por minuto (1 : (625000 : 60)) x 1.000.000. Obs. $098968_{H} = 625000_{10}$
00	Delta-time = 0
FF 2F 00	Fim de track

Cabeçalho do segundo track – canal 0 - piano

4D 54 72 6B	MTrk
00 00 00 19	seguem 25 bytes do presente track (todos os bytes até FF2F00)
00	Delta-time = 0
C0 00	C = Mudar instrumento; 0 = Canal 0; 00 = Piano
00	Delta-time = 0
90	Ativar nota no canal 0
3C 50	3C =Nota Dó5 ; 50_{H} = volume = 80_{10}
60	Delta-time = duas colcheias = uma semínima
3C 00	Como não foi enviado uma mensagem de ativar ou desativar nota, segue o status do evento anterior (running status), ou seja, 90 -> ativar nota no canal 0. 3C 00 significa, portanto, ativar nota Dó5 com um volume igual a 0, o que equivale a desativar a nota Dó5
00	Delta-time = 0
40 50	Running status(90): ativar nota Mi5 (40) com volume 80_{10}
8140	Delta-time de uma mínima (2 semínimas) = 192. 192 : 128 = 1 inteiro e sobra 64_{10} = 40H-> dois bytes de delta-time com o bit mais significativo do primeiro byte = 1. Primeiro byte = 1000 0001 e segundo byte = 0100 0000 => Delta-time = 81 40
40 00	Running status(90): ativar nota Mi3 com volume zero = desativar nota Mi5
00	Delta-time = 0
3C 50	Running status (90) = ativar nota Dó5 com volume 80_{10}
60	Delta-time = uma semínima
3C 00	Running status (90) = ativar nota Dó5 com volume zero = desativar nota Dó5
00	Delta-time = zero
FF 2F 00	Fim do track

Cabeçalho do terceiro track – canal 1 - violão

4D 54 72 6B	MTrk
00 00 00 13	seguem 19 bytes do track

00	Delta-time = 0
C1 18	C = Mudar instrumento; 1 = Canal 1; 18 = Violão de Nylon
60	Delta-time de uma semínima
91	ativar nota no canal 1
3C 50	Nota Dó3, volume 80_{10}
30	Delta-time de uma colcheia
3C 00	running status (91) = ativar nota Dó4 com volume zero = desativar nota Dó4
00	Delta-time = 0
40 50	Running status (91) = ativar Nota Mi4 com volume 80_{10}
30	Delta-time de uma colcheia
40 00	Rrunning status (91) = ativar nota Mi4 com volume zero = desativar nota Mi4
00	Delta-time = 0
FF 2F 00	Fim do track

Capítulo 4

4.1. Conceitos e Notação Musical

Este capítulo tem por objetivo apresentar os termos musicais fundamentais, para explicar o funcionamento de uma máquina MIDI e os formatos utilizados por ela para transmitir e executar músicas seqüenciadas (produzidas por um instrumento eletrônico).

A notação musical tradicional **CPN** – Commom Practice Notation, é uma tentativa de reproduzir, de uma forma gráfica, um conjunto de eventos contínuos no tempo. Esta tentativa faz com que, quando um sistema automático de reprodução de música, baseado nesta forma gráfica, for executar uma determinada obra musical, o resultado seja parecido com uma música executada por uma caixinha de música. Tal fato é amenizado colocando alguns comentários nas partituras, dando uma noção de como a dinâmica dela deve ser executada, ou seja, o quanto se deve afastar dos eventos notados em partitura para se alcançar um resultado mais próximo do original. Desta forma, expressões como *fortíssimo*, *allegro*, etc. são normalmente utilizadas pelos músicos nessas notações.

4.2. Definições de Termos Musicais

4.2.1. Notas Musicais

Nota musical é definida pelo número de repetições de uma forma de onda do espectro sonoro audível em um segundo. Musicalmente falando, é um sinal gráfico

que representa a altura e a duração dos sons musicais. O número de repetições de uma forma de onda por segundo é definido como freqüência do sinal, que é dada em Hertz (Hz). A relação entre a freqüência de uma nota musical e a próxima nota na escala temperada é dada por:

$$\sqrt[12]{2} = 2^{1/12} \approx 1{,}05946309435$$

Como exemplo, pode-se calcular a freqüência da nota Lá# partindo da freqüência conhecida da nota Lá utilizada no diapasão, isto é, padrão de afinação dos instrumentos temperados. Assim, se a nota Lá5 do diapasão é igual a 440 Hz, tem-se que o Lá# é igual a 440 x 1,05946309435 = 446,164Hz.

A figura seguinte ilustra um teclado de piano com as nomenclaturas utilizadas para grafar as notas musicais.

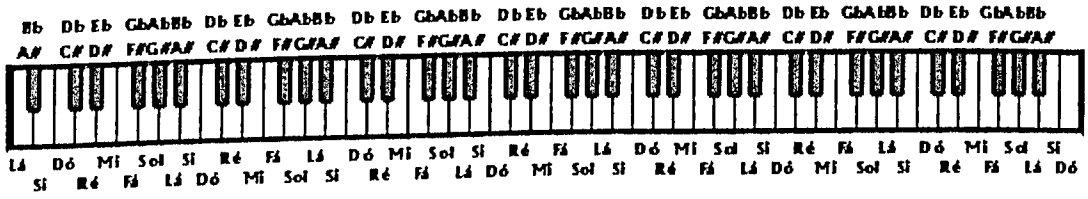


Figura 48

Em que: **Lá = A, Si = B, Dó = C, Ré = D, Mi = E, Fá = F e Sol = G**

Este piano tem como nota mais grave (do lado esquerdo) a nota Lá, que será denominada de Lá0. A nota mais aguda é a nota Dó8. Denomina-se de tessitura do instrumento a extensão das notas que ele consegue reproduzir. Assim, a tessitura de um piano normal vai da nota Lá0 à nota Dó8.

As notas musicais são: Dó, Ré, Mi, Fá, Sol, Lá e Si [2]. Só que entre estas notas existem outras e os seus nomes podem mudar da seguinte maneira: começando pela nota Dó, a próxima pode ser a nota Do# (Dó sustenido) ou a nota Reb (Ré bemol), ou seja, a nota intermediária pode ter o nome da nota anterior acrescida do símbolo # (sustenido), ou o nome da nota posterior acrescida do símbolo *b* (bemol).

Seguindo este raciocínio, pode-se encontrar 12 notas musicais: Dó, Dó# ou Réb, Ré, Ré# ou Mib, Mi, Fá, Fá# ou Solb, Sol, Sol# ou Láb, Lá, Lá# ou Sib, Si. A partir

deste ponto as notas vão se repetindo, mudando apenas a sua freqüência. Vale ainda observar que não apareceu a nota Mi#, pois esta é nota Fá, Fáb é a nota Mi, Si# é a nota Dó e Dób é a nota Si. Ao conjunto das doze notas, principais e intermediárias, denomina-se de oitava. Assim, a cada oitava acima, a freqüência de uma dada nota musical dobra e, descendo uma oitava, a freqüência da mesma nota cai pela metade. A figura seguinte ilustra o que foi dito.

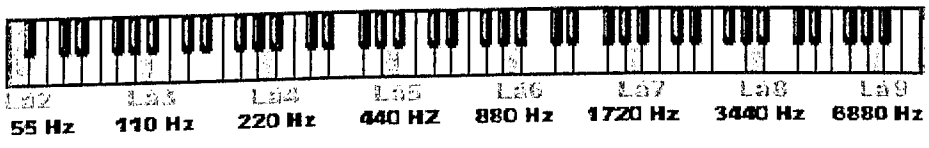


Figura 49

Resumindo, quando a nota possui # (sustenido), significa que ela está localizada meio-tom acima da nota de mesmo nome, e *b* (bemol) significa meio-tom abaixo da nota de mesmo nome [2] [15] [42].

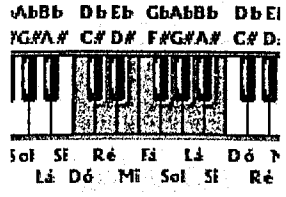


Figura 50

A figura 50 mostra em destaque uma oitava musical, no piano, com as duas nomenclaturas com os respectivos bemóis e sustenidos.

4.2.2. Figuras Musicais

Cada nota musical pode soar um certo intervalo de tempo, dependendo da caixa de ressonância do instrumento. Em uma notação musical, os tempos de cada nota são múltiplos uns dos outros. Assim, pode-se apresentar esta relação conforme tabela seguinte:

Figura e Pausa	Código	Nome em Português	Nome em Inglês	Valor Proporcional
	1	Semibreve	Hole Note	1
	2	Mínima	Half Note	½
	4	Semínima	Quarter Note	¼
	8	Colcheia	Eigth Note	$^1/_8$

	16	Semicolcheia	16th Note	$^1/_{16}$
	32	Fusa	32th Note	$^1/_{32}$
	64	Semifusa	64th Note	$^1/_{64}$

Tabela 18

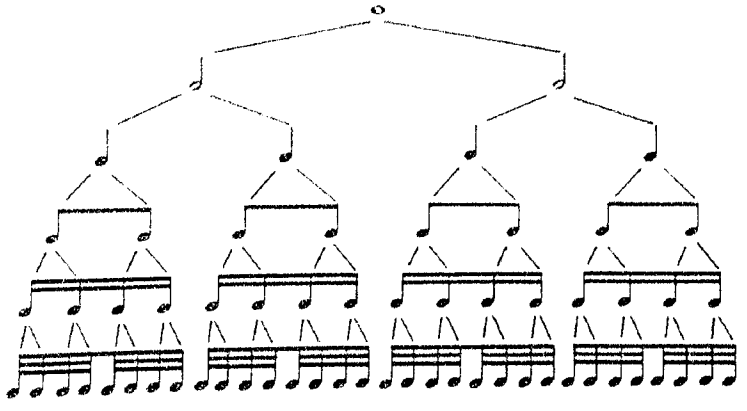


Figura 51

4.2.3. Acidentes Musicais

Também chamados de alterações musicais, pois alteram a nota em sua posição, freqüência original.

Símbolo	Nome	Efeito
♯	Sustenido	Eleva a nota meio-tom.
𝄪	Dobrado Sustenido	Equivale a dois sustenidos, ou seja, eleva a nota um tom.
♭	Bemol	Abaixa a nota meio-tom.
𝄫	Dobrado Bemol	Equivale a dois bemóis, ou seja, abaixa a nota um tom.
♮	Bequadro	Anula o efeito dos demais acidentes.

Tabela 19

Se na frente de uma nota colocar-se um ponto, isto significa que o valor da nota é aumentado pela metade. Assim, uma semínima pontuada (♩.) possui o valor de uma semínima mais uma colcheia. A figura 51 ilustra o que foi dito.

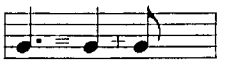


Figura 52

4.2.4. Claves Musicais

São símbolos colocados na extremidade esquerda da pauta, que servem para determinar o nome e a altura das notas correspondentes a cada uma das linhas e espaços da pauta. As claves são as seguintes:

Símbolo	Nome	Função
𝄞	Clave de Sol	Usada para instrumentos agudos.
𝄢	Clave de Fá	Usada para instrumentos graves.
𝄡	Clave de Dó	Usada para instrumentos de sons intermediários agudos.
𝄥	Clave de Percussão	Usada para instrumentos de percussão.

Tabela 20

Existem diferentes claves para facilitar a leitura musical nos diversos instrumentos existentes. O violino, por exemplo, usa a clave de sol, já o violoncelo, a clave de fá. O piano possui uma particularidade em relação às claves. Para abranger todas as notas do piano, são necessárias duas claves: clave de Sol e clave de Fá.

4.2.4.1. Distribuição das Notas na Clave de Sol e de Fá

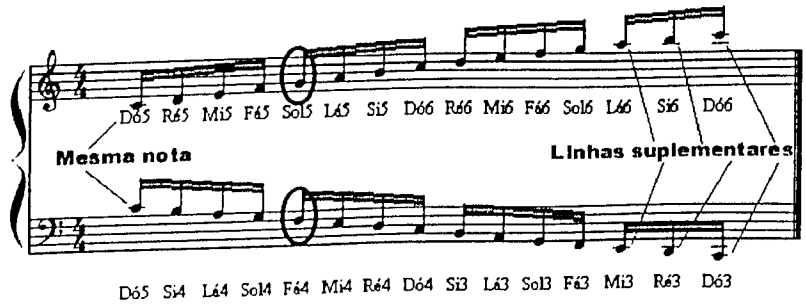


Figura 53

Pode-se perceber que o Sol5 e o Fá4 estão escritos na linha em que começam os desenhos das Claves. Na linha em que o desenho da clave inicia, localiza-se a nota que levará o nome da clave. Se a clave de sol começa na 2ª linha, logo a nota localizada na

2ª linha é a nota sol. As linhas suplementares servem para escrever as notas que estão localizadas acima ou abaixo do pentagrama. Assim, em vez de mais uma linha, tem-se um tracinho [2] [15] [42].

4.2.5. Fórmula de Compasso (Time Signature)

Entende-se por compasso a medida musical usada para dividir uma música em fragmentos de igual duração. Os compassos podem possuir diversos tipos de divisão, que são representadas por números parecidos com frações matemáticas. 2/4, 3/4, 6/8, 9/8, etc., que são as fórmulas de compasso [15] [42].

Fórmula de compasso, portanto, são os dois números localizados à direita da Armadura de Clave, que servem para indicar a maneira como o compasso será preenchido.

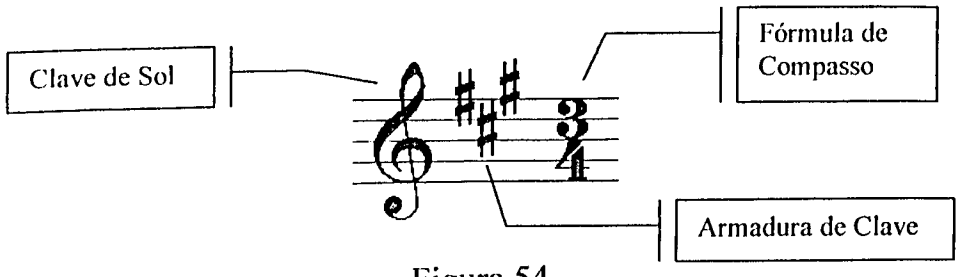


Figura 54

Os compassos dividem-se basicamente em duas categorias: simples e composto. Se o número superior for 2, 3 ou 4, então o compasso será simples. Mas se o número superior for 6, 9 ou 12, então o compasso será composto.

4.2.5.1. Exemplos de Compassos Simples

No compasso simples o número superior indica a quantidade de notas que cabe em um compasso, e o número inferior indica o código da figura musical, ou também chamada de unidade de tempo. Por exemplo: 2/4 quer dizer que são necessárias duas figuras de código 4 (semínima, ou quarter note) para preencher o compasso.

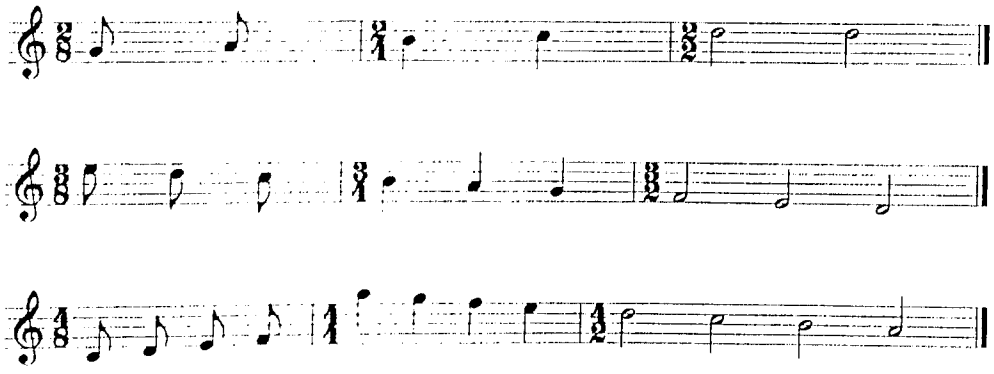


Figura 55

4..2.5.2. Exemplos de Compassos Compostos

No compasso composto o número superior indica a quantidade de notas que cabe em um compasso, e o número inferior indica o tipo de figura musical, porém não indica a unidade de tempo. Por exemplo: 6/8 quer dizer que são necessárias seis figuras do tipo 8 (colcheia, ou eighth note) para preencher o compasso.

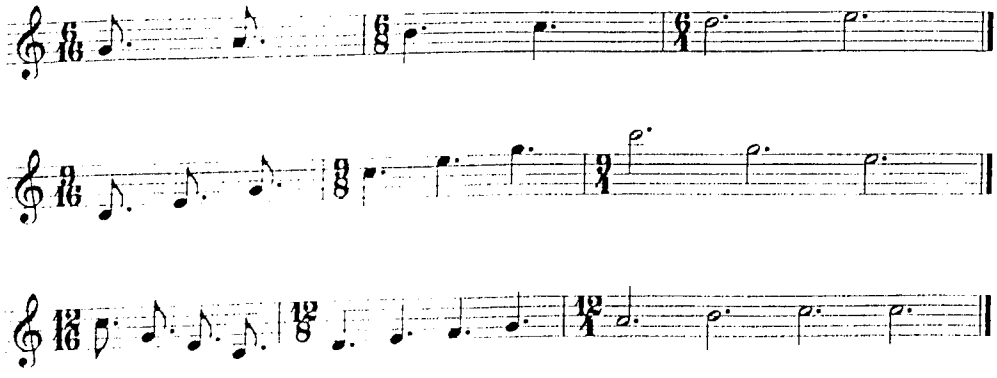


Figura 56

4..2.6. Armadura de Clave (Key Signature)

É um conjunto de acidentes (# e *b*), colocados logo após a clave, tendo como finalidade indicar a tonalidade da música. Esses acidentes obedecem a uma ordem pré-definida. [15] [42]

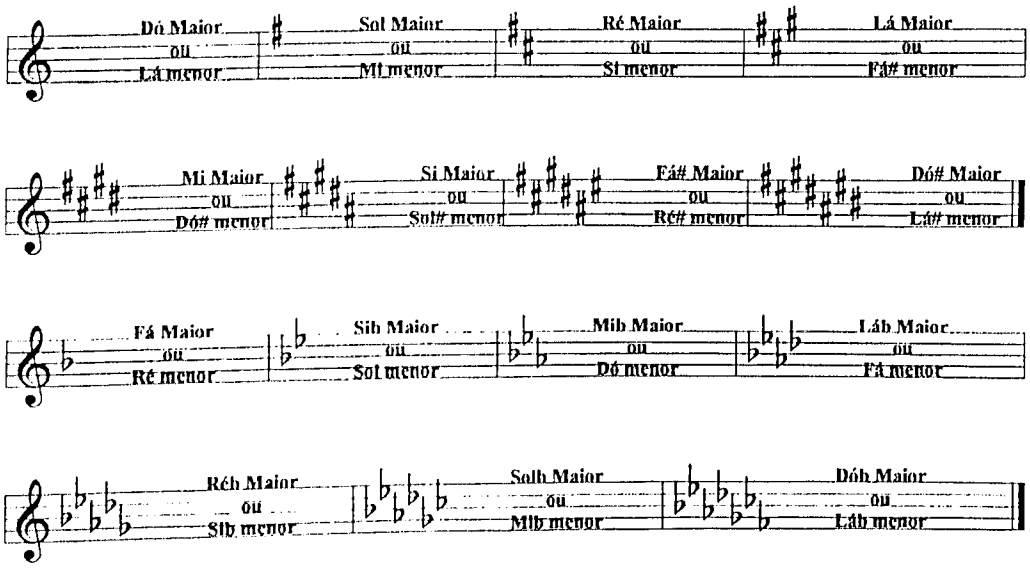


Figura 57

Esses acidentes indicam que todas as notas da música que coincidam com o acidente grafado na armadura serão ou bemóis ou sustenidos, seja qual for sua altura (freqüência da nota), salvo indicação ao contrário cancelando o acidente. Isto evita a poluição de acidentes na notação musical.

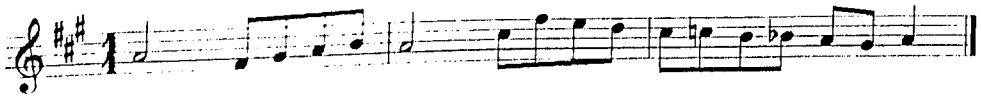


Figura 58

A figura anterior mostra uma armadura de clave com acidentes nas notas Fá, Dó e Sol. Isto indica que todas as notas Fá, Dó e Sol do pentagrama, seja qual for sua altura, serão Fá#, Dó# e Sol#. Mas se alguma nota for precedida do símbolo de bequadro (♮), será uma nota natural. Neste exemplo a 2ª nota do 3° compasso é um Dó natural (sem acidente).

4..2.7. Tonalidade Maior

As escalas possuem 7 notas, que serão aqui localizadas no campo cinza. Para monta-las, basta colocar as notas nas posições indicadas no campo cinza com os números em negrito.

1ª	1ª#/2ªb	**2ª**	2ª#/3ªb	**3ª**	**4ª**	4ª#/5ªb	**5ª**	5ª#/6ªb	**6ª**	6ª#/7ªb	**7ª**

Montando a escala de Dó Maior, ou seja, começando a colocar notas nas posições indicadas em cinza com os números em negrito, a partir da nota Dó, tem-se o seguinte:

Dó M (Dó Maior)

Dó		**Ré**		**Mi**	**Fá**		**Sol**		**Lá**		**Si**

Mi M (Mi Maior)

Mi		**Fá#**		**Sol#**	**Lá**		**Si**		**Dó#**		**Ré#**

Sol M (Sol Maior)

Sol		**Lá**		**Si**	**Dó**		**Ré**		**Mi**		**Fá#**

4.2.8. Acorde

Acorde é um conjunto de no mínimo três notas, formado com a 1ª, 3ª e 5ª notas de cada escala, podendo ser acrescido ou não das outras notas da escala.

Montando-se acordes de três notas sobre cada nota da escala de DoM, tem-se o que é chamado de Campo Harmônico de Dó Maior, ficando os acordes da seguinte maneira [5]:

1ª nota:	Dó	**3ª nota:**	Mi	**5ª nota:**	Sol	=	acorde de Dó maior
1ª nota:	Ré	**3ª nota:**	Fá	**5ª nota:**	Lá	=	acorde de Ré menor
1ª nota:	Mi	**3ª nota:**	Sol	**5ª nota:**	Si	=	acorde de Mi menor
1ª nota:	Fá	**3ª nota:**	Lá	**5ª nota:**	Dó	=	acorde de Fá maior
1ª nota:	Sol	**3ª nota:**	Si	**5ª nota:**	Ré	=	acorde de Sol maior
1ª nota:	Lá	**3ª nota:**	Dó	**5ª nota:**	Mi	=	acorde de Lá menor
1ª nota:	Si	**3ª nota:**	Re	**5ª nota:**	Fá	=	acorde de Si diminuto

Tabela 21

Montar os acordes é bastante simples, resta porém, classificá-los para saber que acorde é Maior, Menor, Diminuto, Aumentado, etc.

Tipo de acorde	1ª nota da escala	3ª nota da escala	5ª nota da escala
Maior	1ª nota	4 semitons acima da 1ª nota	3 semitons acima da 3ª nota
menor	1ª nota	3 semitons acima da 1ª nota	4 semitons acima da 3ª nota
Diminuto	1ª nota	3 semitons acima da 1ª nota	3 semitons acima da 3ª nota
Aumentado	1ª nota	4 semitons acima da 1ª nota	4 semitons acima da 3ª nota

Tabela 22

Deve-se tomar cuidado para não misturar as tonalidades que possuem # com as que possuem *b*. Das tonalidades maiores que possuem *b*, a única que não leva um *b* no nome é Fá Maior. E a única que não tem nenhum # nem b é a tonalidade de Dó Maior. Os acordes são representados também por abreviações chamadas de cifras que possuem várias combinações para poder representar todos os possíveis acordes [5].

4.2.8.1. Cifras

- A = Lá Maior
- B = Si Maior
- C = Dó Maior
- D = Ré Maior
- E = Mi Maior
- F = Fá Maior
- G = Sol Maior
- M = Maior
- m = menor
- 7 = com sétima
- º = diminuto
- # = Sustenido
- *b* = bemol

4.2.8.2. Exemplos:

- A7 = Lá com sétima
- Am7 = Lá menor com sétima
- G7M = Sol com Sétima Maior
- Bº = Sí Diminuto
- F#m = Fá Sustenido menor
- Eb = Mi Bemol Maior

4.2.8.3. Outras abreviações:

- M = Maior
- m = Menor
- aum. ou a = Aumentado
- dim. ou d, ou º = diminuto
- # = Sustenido
- *b* = Bemol

4.2.9. Intervalos

O intervalo é a distância entre dois sons, podendo ser: simples ou composto, ascendente ou descendente, melódico ou harmônico [15] [42].

4.2.9.1. Simples: não ultrapassa uma oitava.

4.2.9.2. Composto: ultrapassa uma oitava.

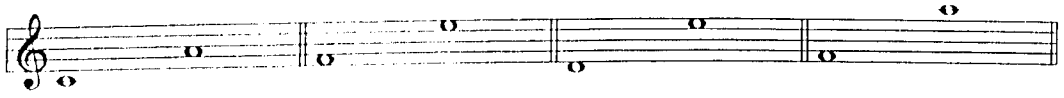


Figura 59

4.2.9.3. Ascendente: o 1° som é mais grave que o 2°.

4.2.9.4. Descendente: o 1° som é mais agudo que o 2°.

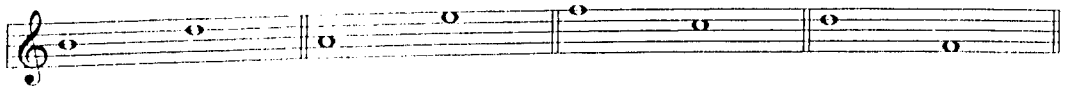


Figura 60

4.2.9.5. Melódico: sons consecutivos.

4.2.9.6. Harmônico: sons simultâneos.

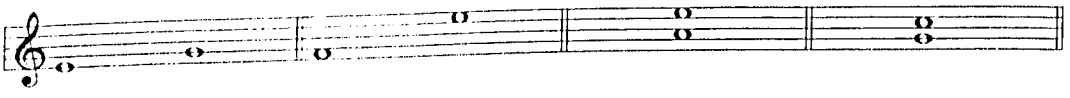


Figura 61

4.2.9.7. Classificação dos intervalos usando como referência a escala de Dó Maior:

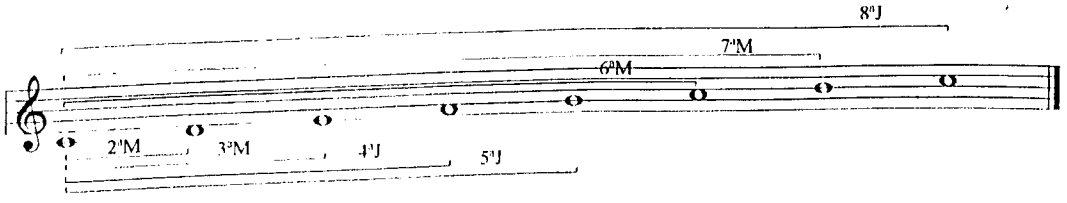


Figura 62

Na escala de Dó Maior encontram-se os seguintes intervalos: 2ªM, 3ªM, 4ªJ, 5ªJ, 6ªM, 7ªM, 8ªJ (todos em relação à tônica – I grau). Partindo-se destes intervalos é possível encontrar os demais, ficando da seguinte maneira:

- M – 1 st. = m
- M + 1 st. = aum.
- m + 1 st. = M
- m – 1 st. = dim.
- J + 1 st. = aum.
- J – 1 st. = dim.

4.2.9.8. Abreviações:

- M = Maior
- m = Menor
- aum. = Aumentado
- dim. = Diminuto
- m – 1 st. = dim.
- J + 1 st. = aum.
- J – 1 st. = dim

4.2.9.9. Resumindo:

- 2^a m = 1/2 tom
- 2^a M = 1 tom
- 3^a m = 1 tom e 1/2
- 3^a M = 2 tons
- 4^a J = 2 tons e 1/2
- 4^a aum. ou 5^a dim. = 3 tons
- 5^a J = 3 tons e 1/2
- 5^a aum. ou 6^a m = 4 tons
- 6^a M ou 7^a dim. = 4 tons e 1/2
- 7^a m = 5 tons
- 7^a M = 5 tons e 1/2

4.2.10. Tonalidades Menores

As escalas possuem 7 notas, que serão aqui localizadas no campo cinza. Para monta-las, basta colocar as notas nas posições indicadas no campo cinza com os números em negrito.

4.2.10.1. Escala Menor Natural

1ª	1ª#/2ªb	**2ª**	**3ª**	3ª#/4ªb	**4ª**	4ª#/5ªb	**5ª**	**6ª**	6ª#/7ªb	**7ª**	7ª#

Montando a escala de Lám natural, ou seja, começando a colocar notas nas posições indicadas em cinza com os números em negrito a partir da nota Lá, tem-se o seguinte:

Lám (Lá menor - Natural)

Lá		**Si**	**Dó**		**Ré**		**Mi**	**Fá**		**Sol**	

Sim (Si menor - Natural)

Si		**Dó#**	**Ré**		**Mi**		**Fá#**	**Sol**		**Lá**	

4.2.10.2. Escala Menor Harmônica

A escala menor harmônica possui a 7ª nota elevada meio-tom.

1ª	1ª#/2ªb	**2ª**	**3ª**	3ª#/4ªb	**4ª**	4ª#/5ªb	**5ª**	**6ª**	6ª#7ªb	7ª	**7ª#**

Lám (Lá menor - Harmônica)

Lá		**Si**	**Dó**		**Ré**		**Mi**	**Fá**			**Sol#**

Sim (Si menor - Harmônica)

Si		**Dó#**	**Ré**		**Mi**		**Fá#**	**Sol**			**Lá#**

4.2.10.3. Escala Menor Melódica

A escala Menor Melódica possui a 6ª e a 7ª notas elevadas meio-tom.

1ª	1ª#/2ªb	**2ª**	**3ª**	3ª#/4b	**4ª**	4#/4b	**5ª**	6ª	**6ª#**	7ª	**7ª#**

Lám (Lá menor - Melódica)

Lá		**Si**	**Dó**		**Ré**		**Mi**		**Fá#**		**Sol#**

Sim (Si menor - Melódica)

Si		**Dó**	**Ré**		**Mi**		**Fá#**		**Sol#**		**Lá#**

4.2.10.4. Os acordes de quatro notas montados sobre as notas das escalas:

Nota	Escala Maior	Escala Menor Natural	Escala Menor Harmônica	Escala Menor Melódica
1ª nota	7M	m7	m7M	m7M
2ª nota	m7	m7(b5)	m7(b5)	m7
3ª nota	m7	7M	7M(#5)	7M(#5)
4ª nota	7M	m7	m7	7
5ª nota	7	m7	7	7
6ª nota	m7	7M	7M	m7(b5)
7ª nota	m7(b5)	7	Dim	m7(b5)

Tabela 23

As notas da escala também podem ser chamadas de graus, sendo representados por algarismos romanos, ou seja, 1ª nota I (1º grau), 2ª nota II (2º grau)...., 5ª nota V (5º grau), etc.

Os acordes podem começar com qualquer nota do seu grupo de notas sem no entanto mudar o seu nome (com exceção dos acordes diminutos que, ao mudar a nota mais grave, esta passa a dar o nome do acorde). Quando isto acontecer, o acorde será nomeado da seguinte maneira: C7M possui as notas: do, mi, sol, si. Se ele começasse, por exemplo, com a nota mi, mas tivesse todas as outras notas do acorde de C7M, o acorde iria se chamar C7M/E, indicando que a nota localizada depois da barra / passou a ser a nota mais grave do acorde. Lê-se Do com 7ª Maior e Mi no baixo. Logo, o acorde continua com o mesmo nome, mudando apenas a nota mais grave, ou seja, a nota que fica depois da barra / .

Não importa qual seja o Campo Harmônico, porque os acordes obedecem a uma ordem de prioridade:

- O mais importante é o 1º grau, ou seja, o acorde formado sobre a 1ª nota da escala.
- Em 2º lugar está o 5º grau, ou seja, o acorde formado sobre a 5ª nota da escala.
- Em 3º lugar está o 4º grau, ou seja, o acorde formado sobre a 4ª nota da escala.

Capítulo 5

5.1. Projeto e Implementação das Rotinas de Manipulação de Arquivos MIDI SMF Formato 1 sem Running Status.

Esta parte da pesquisa procurará identificar e quantificar corretamente as notas musicais e seus parâmetros elementares dentro de um arquivo MIDI SMF formato 1. Como um dos objetivos deste estudo é apresentar formas de manipular os arquivos MIDI SMF formato 1 no paradigma funcional e elaborar ferramentas básicas de trabalho, adotar-se-á a metodologia de desenvolvimento passo a passo das rotinas básicas que manipularão o protocolo. [23] [24] [25] [26]

Dada a partitura:

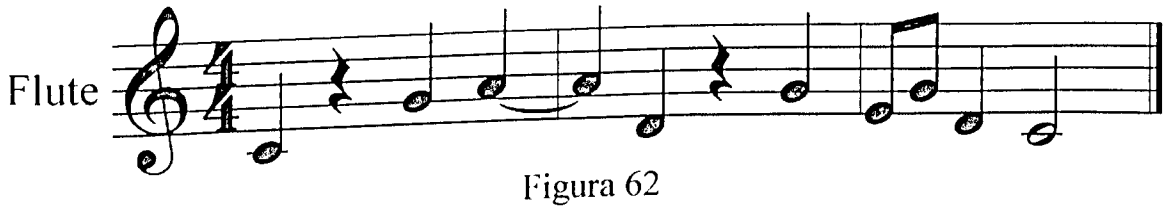


Figura 62

O Arquivo MIDI formato 1, em hexadecimal correspondente é:

4D 54 68 64 00 00 00 06 00 01 00 02 00 F0 4D 54 72 6B 00 00 00 13 00 FF 58 04 04 02 18 08 00 FF 51 03 09 27 C0 00 FF 2F 00 4D 54 72 6B 00 00 00 55 00 90 3C 50 81 70 80 3C 40 81 70 90 43 50 81 70 80 43 40 00 90 45 50 83 60 80 45 40 00 90 3E 50 81 70 80 3E 40 81 70 90 43 50 81 70 80 43 40 00 90 40 50 78 80 40 40 00 90 43 50 78 80 43 40 00 90 3E 50 81 70 80 3E 40 00 90 3C 50 83 60 80 3C 40 00 FF 2F 00

Esta notação em hexa é a mais utilizada pelos profissionais em MIDI [53], porém, a manipulação dos dados em decimal evita uma perda de tempo de processamento na conversão de decimal para hexa. O arquivo, em decimal, fica:

77, 84, 104, 100, 0, 0, 0, 6, 0, 1, 0, 2, 4, 0, 77, 84, 114, 107, 0, 0, 0, 26, 0, 255, 88, 4, 4, 2, 24, 8, 0, 255, 89, 2, 0, 0, 0, 255, 81, 3, 7, 161, 32, 224, 0, 255, 47, 0, 77, 84, 114, 107, 0, 0, 0, 99, 0, 255, 3, 5, 70, 108, 117, 116, 101, 0, 192, 73, 0, 144, 60, 64, 136, 0, 128, 60, 0, 136, 0, 144, 67, 64, 136, 0, 128, 67, 0, 0, 144, 69, 64, 144, 0, 128, 69, 0, 0, 144, 62, 64, 136, 0, 128, 62, 0, 136, 0, 144, 67, 64, 136, 0, 128, 67, 0, 0, 144, 64, 64, 132, 0, 128, 64, 0, 0, 144, 67, 64, 132, 0, 128, 67, 0, 0, 144, 62, 64, 136, 0, 128, 62, 0, 0, 144, 60, 64, 144, 0, 128, 60, 0, 0, 255, 47, 0

Conforme visto no capítulo sobre MIDI, o arquivo MIDI SMF 1 possui um track principal, contendo o cabeçalho e parâmetros básicos do protocolo, e um ou mais tracks contendo as informações da música, pois este estudo visa capacitar um usuário de informática, ou um músico com tais conhecimentos, implementar programas de manipulação de arquivos MIDI formato 1. Desta forma, pode-se dividir a funções que se seguem em três grandes grupos:

5.1.1 Leitura do arquivo MIDI e conversão do mesmo em uma lista de inteiros para posterior manipulação, tratamento e extração dos parâmetros básicos do cabeçalho principal. Neste grupo portanto, serão implementadas funções para:

- Ler o arquivo MIDI e convertê-lo em uma lista de inteiros;
- Extrair o track principal do arquivo MIDI;
- Leitura do formato do arquivo SMF;
- Leitura do número de tracks;
- Leitura do valor da ppq.

5.1.1.1. Ler o arquivo MIDI e convertê-lo em uma lista de inteiros

Esta rotina é a básica para o início da implementação das demais ferramentas. Como existe uma função primitiva no CLEAN que converte binário para inteiro, para uma maior velocidade de processamento, optou-se pela geração de uma lista de inteiros como resultado da leitura do arquivo MIDI.

Figura 63

O programa:

listaTracks.icl - D:\ANDRE\Tese\Programas\listaTracks.icl

```
1 module listaTracks
2 import StdEnv, StdIO
3
4 Start comp
5 //para abrir o explorer
6 #    (maybeFile, comp)= selectInputFile comp
7 #    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
8 //fim da abertura do explorer
9 #    (ok, file, comp)= fopen nomeArquivo FReadText comp
10 #    (conteudo, file)= freads file 300000
11 #    (ok, comp)= fclose file comp
12 #listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
13 = listaTracks
```

Figura 64

Resultado obtido:

[77, 84, 104, 100, 0, 0, 0, 6, 0, 1, 0, 2, 4, 0, 77, 84, 114, 107, 0, 0, 0, 26, 0, 255, 88, 4, 4, 2, 24, 8, 0, 255, 89, 2, 0, 0, 0, 255, 81, 3, 7, 161, 32, 224, 0, 255, 47, 0, 77, 84, 114, 107, 0, 0, 0, 99, 0, 255, 3, 5, 70, 108, 117, 116, 101, 0, 192, 73, 0, 144, 60, 64, 136, 0, 128, 60, 0, 136, 0, 144, 67, 64, 136, 0, 12, 8, 67, 0, 0, 144, 69, 64, 144, 0, 128, 69, 0, 0, 144, 62, 64, 136, 0, 128, 62, 0, 136, 0, 144, 67, 64, 136, 0, 128, 67, 0, 0, 144, 64, 64, 132, 0, 128, 64, 0, 0, 144, 67, 64, 132, 0, 128, 67, 0, 0, 144, 62, 64, 13, 6, 0, 128, 62, 0, 0, 144, 60, 64, 144, 0, 128, 60, 0, 0, 255, 47, 0]

Esta rotina devolve como resposta uma única lista contendo todos os eventos do arquivo MIDI .

5.1.1.2. Extrair o track principal[34] do arquivo MIDI

De posse do arquivo MIDI convertido para uma lista de inteiros, função implementada no item I, pode-se extrair da mesma o track principal[35].

Figura 65

O programa:

mThd.icl - D:\ANDRE\Tese\Programas\mThd.icl

```
1 module mThd
2 import StdEnv, StdIO
3
4 Start comp
5 //para abrir o explorer
6 #    (maybeFile, comp)= selectInputFile comp
7 #    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
8 //fim da abertura do explorer
9 #    (ok, file, comp)= fopen nomeArquivo FReadText comp
10 #    (conteudo, file)= freads file 300000
11 #    (ok, comp)= fclose file comp
12 #listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
13 #mThd =: take 14 listaTracks
14 = mThd
```

Figura 66

Resultado obtido:

[77,84,104,100,0,0,0,6,0,1,0,2,4,0]

77, 84, 104, 100 = Mthd

0,0,0,6 = Tamanho do cabeçalho

0, 1 = Formato 1

0, 2 = Número de Tracks

4, 0 = ppq

[34] O primeiro track possui 14 bytes de informação contendo o cabeçalho principal com o tipo de formato, número de tracks e a ppq.

[35] Por motivos didáticos, adotou-se, primeiro, construir uma lista de inteiros do arquivo MIDI para, posteriormente, separar os tracks e retirar as informações necessárias.

5.1.1.3. Leitura do formato do arquivo SMF

A informação do formato está registrada nos bytes 9° e 10° do Track principal (em **negrito**), sendo que o formato é calculado da seguinte forma:

Figura 67

Track Principal = [77,84,104,100,0,0,0,6,**0,1**,0,2,4,0]

Formato = (Byte mais significativo(9°)*256) + (Byte menos significativo(10°))

O programa:

formato.icl - D:\ANDRE\Tese\Programas\formato.icl

```
1 module formato
2 import StdEnv, StdIO
3
4 Start comp
5 //para abrir o explorer
6 #    (maybeFile, comp)= selectInputFile comp
7 #    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
8 //fim da abertura do explorer
9 #    (ok, file, comp)= fopen nomeArquivo FReadText comp
10 #    (conteudo, file)= freads file 300000
11 #    (ok, comp)= fclose file comp
12 #listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
13 #formato =: listaTracks !! 9
14 = formato
```

Figura 68

Resultado obtido:

1

5.1.1.4. Leitura do número de tracks[36]

A informação do número de tracks está registrada nos bytes 11° e 12° (em **negrito**) do track do cabeçalho principal.

[36] O número de tracks disponibilizado no cabeçalho principal indica quantos tracks existem após o track principal.

Figura 69

Track principal = [77,84,104,100,0,0,0,6,0,1,**0,2**,4,0]

O número de tracks é calculado da seguinte forma:

NTracks = (Byte mais significativo(11°)*256) + (Byte menos significativo(12°))

O programa:

nTracks.icl - D:\ANDRE\Tese\Programas\nTracks.icl

```
1 module nTracks
2 import StdEnv, StdIO
3
4 Start comp
5 //para abrir o explorer
6 #    (maybeFile, comp)= selectInputFile comp
7 #    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
8 //fim da abertura do explorer
9 #    (ok, file, comp)= fopen nomeArquivo FReadText comp
10 #    (conteudo, file)= freads file 300000
11 #    (ok, comp)= fclose file comp
12 #listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
13 #mThd =: take 14 listaTracks
14 #nTracks =: (mThd!!10)*256 + (mThd!!11)
15 = nTracks
```

Figura 70

Resultado obtido:

2

5.1.1.5. Leitura do valor da ppq

A informação da **ppq** está registrada nos bytes 13° e 14° (**em negrito**).

Figura 71

Track Principal = [77,84,104,100,0,0,0,6,0,1,0,2,**4,0**]

A **ppq** é calculada da seguinte forma:

ppq = (Byte mais significativo(13^{O})*256) + (Byte menos significativo(14^{O}))

O programa:

ppq.icl - D:\ANDRE\Tese\Programas\ppq.icl

```
1 module ppq
2 import StdEnv, StdIO
3
4 Start comp
5 //para abrir o explorer
6 #    (maybeFile, comp)= selectInputFile comp
7 #    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
8 //fim da abertura do explorer
9 #    (ok, file, comp)= fopen nomeArquivo FReadText comp
10 #    (conteudo, file)= freads file 300000
11 #    (ok, comp)= fclose file comp
12 #listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
13 #ppq =: ((listaTracks !! 12)*256) + (listaTracks !! 13)
14 = ppq
```

Figura 72

Resultado obtido:

1024

5.1.2. Ferramentas básicas gerais: estas ferramentas, isto é, funções, deverão trabalhar os arquivos MIDI em várias camadas. Cada função intermediária poderá ser utilizada para implementações de novas ferramentas. Dentre estas funções, destacam-se as de leitura dos Meta-Eventos existentes em cada track e suas conversões para o domínio musical. Neste grupo portanto, serão implementadas as seguintes funções:

- Criando uma lista dos mTracks,
- Gerando uma lista de tracks,
  - Pegando o conteúdo de um track,
- Separando todos os tracks de um arquivo MIDI,
- Eliminando o cabeçalho e número de bytes de cada track mTrk,
- Separando os eventos dos tracks,
  - Primeiro Track,
  - Convertendo Delta Times para ppq,
    - Primeiro byte do Delta Time menor que 80_{H} (128_{10}),
    - Primeiro byte do delta time maior que 7F (127_{10}) e o segundo menor ou igual a 7F (127_{10}),
  - Separando os eventos dos demais tracks,
- Conversão de ppq em Delta Time,
- Metrônomo,
- Tonalidade da música,
- Fórmula de compasso,
- Instrumento do canal.

5.1.2.1. Criando uma lista dos mTracks

Esta rotina devolve uma lista de todos os tracks, sem o track do cabeçalho principal, ou seja, sem os 14 primeiros bytes.

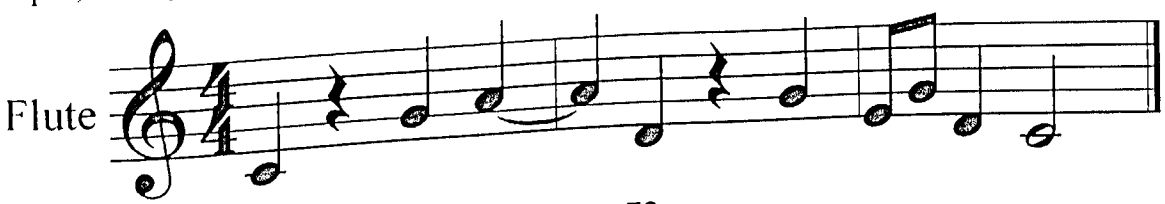


Figura 73

Track Principal = [77,84,104,100,0,0,0,6,0,1,0,2,4,0]

O programa:

mTrks.icl - D:\ANDRE\Tese\Programas\mTrks.icl

```
1 module mTrks
2 import StdEnv, StdIO
3
4 Start comp
5 //para abrir o explorer
6 #    (maybeFile, comp)= selectInputFile comp
7 #    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
8 //fim da abertura do explorer
9 #    (ok, file, comp)= fopen nomeArquivo FReadText comp
10 #    (conteudo, file)= freads file 300000
11 #    (ok, comp)= fclose file comp
12 #listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
13 #mTrks =: drop 14 listaTracks
14 =mTrks
15
```

Figura 74

Resultado obtido: Uma lista com todos os tracks menos o principal.

[77, 84, 114, 107, 0, 0, 0, 26, 0, 255, 88, 4, 4, 2, 24, 8, 0, 255, 89, 2, 0, 0, 0, 255, 81, 3, 7, 161, 32, 2, 24, 0, 255, 47, 0, 77, 84, 114, 107, 0, 0, 0, 99, 0, 255, 3, 5, 70, 108, 117, 116, 101, 0, 192, 73, 0, 144, 60, 64, 136, 0, 128, 60, 0, 136, 0, 144, 67, 64, 136, 0, 128, 67, 0, 0, 144, 69, 64, 144, 0, 128, 69, 0, 0, 144, 62, 64, 136, 0, 128, 62, 0, 136, 0, 144, 67, 64, 136, 0, 128, 67, 0, 0, 144, 64, 64, 132, 0, 128, 64, 0, 0, 144, 67, 64, 132, 0, 128, 67, 0, 0, 144, 62, 64, 136, 0, 128, 62, 0, 0, 144, 60, 64, 144, 0, 128, 60, 0, 0, 255, 47, 0]

5.1.2.2. Gerando uma lista de tracks

Para separar a lista em tracks, é necessário conhecer o tamanho de cada track. O tamanho do mesmo está registrado do 5º ao 8º byte do track. Para obtê-lo, elimina-se o track principal (mThd), e adicionam-se 8 bytes ao valor calculado no 5º ao 8º byte do track desejado. Estes oito bytes representam o identificador de track (4 Bytes) mais os 4 bytes que registram o tamanho do track (5º ao 8º byte).

tamanhoTrack =: (((mTrks!!4) * 256 * 256 * 256) + ((mTrks!!5) * 256 * 256) + ((mTrks!!6) * 256) + (mTrks!!7) +8)

5.1.2.2.1. Pegando o conteúdo de um track

O programa a seguir, de posse do tamanho do track, cria uma lista contendo apenas este track. Isto basicamente é feito da seguinte forma:

Figura 75

(take tamanhoTrack mTrks)

O PROGRAMA

primeiro.icl - D:\ANDRE\Tese\Programas\primeiro.icl

```
1 module primeiro
2 import StdEnv, StdIO
3
4 Start comp
5 //para abrir o explorer
6 #    (maybeFile, comp)= selectInputFile comp
7 #    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
8 //fim da abertura do explorer
9 #    (ok, file, comp)= fopen nomeArquivo FReadText comp
10 #    (conteudo, file)= freads file 300000
11 #    (ok, comp)= fclose file comp
12 #listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
13 #mTrks =: drop 14 listaTracks // elimina o track principal
14 #tamanhoTrack =: (   ((mTrks!!4)*256*256*256)+((mTrks!!5)*256*256)+
                                    ((mTrks!!6)*256) + (mTrks!!7) +8 )
15
16 #PegaPrimeiromTrk =: take tamanhoTrack mTrks
17 =PegaPrimeiromTrk
```

Figura 76

Resultado: A lista com o primeiro track do exemplo da figura 75

[77, 84, 114, 107, 0, 0, 0, 26, 0, 255, 88, 4, 4, 2, 24, 8, 0, 255, 89, 2, 0, 0, 0, 255, 81, 3, 7, 161, 32, 224, 0, 255, 47, 0]

O próximo passo é implementar um programa que separe todos os tracks do arquivo MIDI, cada um em uma lista.

5.1.2.3. Separando todos os tracks de um arquivo MIDI

De posse da rotina básica que separa um track, para se separar todos os tracks do arquivo basta modificar o programa para pegar o primeiro track da lista contendo o arquivo MIDI sem o track principal e colocá-lo em uma nova lista, a saber, a lista que conterá todas as listas de tracks. Após colocar a lista do primeiro track na lista final, elimina-se a mesma da lista inicial e pega-se o próximo track, transforma-o em uma lista e o coloca na nova lista de tracks, o processo continua até que a lista inicial de tracks retorne vazia. Quando isto ocorrer, a lista estará invertida, já que a implementação a seguir coloca cada track na cabeça da lista. Para finalizar, basta, portanto, inverter a lista obtida.

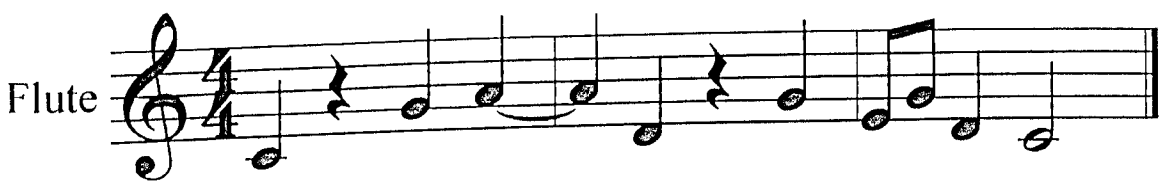


Figura 77

O programa

O programa ficou maior do que a janela, portanto não foi possível captura-la. Assim segue-se o programa em modo texto.

```
module listaTracksMTrks
import StdEnv,StdIO
Start comp
//para abrir o explorer
#    (maybeFile, comp)= selectInputFile comp
#    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
//fim da abertura do explorer
#    (ok, file, comp)= fopen nomeArquivo FReadText comp
#    (conteudo, file)= freads file 300000
#    (ok, comp)= fclose file comp
#listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
#mThd =: take 14 listaTracks
#mTrks =: drop 14 listaTracks
#nTracks =: (mThd!!10)*256 + (mThd!!11)
#listaTracksMTrks =: (reverse (SeparaTracks mTrks [] nTracks))
//reverse inverte a lista, conforme explicação em texto anterior
```

```
=listaTracksMTrks
/*função que pega a lista com o arquivo MIDI e o converte em uma lista com cada
track em uma lista*/
SeparaTracks mTrks mThdMaisMTrks nTracks
     |nTracks == 0 = mThdMaisMTrks
     |otherwise =
      SeparaTracks (drop tamanhoTrack mTrks)( [take (tamanhoTrack)
                    mTrks]++ mThdMaisMTrks) (nTracks -1)
where
 tamanhoTrack =:(  ((mTrks!!4)*256*256*256)+((mTrks!!5)*256*256)+
                    ((mTrks!!6)*256) + (mTrks!!7) +8 )
```

Obs. Para se obter a lista com todos os tracks, basta fazer um **append** do **mThd** com a lista de **tracks mTrks**.

Resultado obtido: Lista contendo listas com todos os tracks do arquivo MIDI.

[[77, 84, 114, 107, 0, 0, 0, 26, 0, 255, 88, 4, 4, 2, 24, 8, 0, 255, 89, 2, 0, 0, 0, 255, 81, 3, 7, 161, 32, 224, 0, 255, 47, 0], [77, 84, 114, 107, 0, 0, 0, 99, 0, 255, 3, 5, 70, 108, 117, 116, 101, 0, 192, 73, 0, 144, 60, 64, 136, 0, 128, 60, 0, 136, 0, 144, 67, 64, 136, 0, 128, 67, 0, 0, 144, 69, 64, 144, 0, 128, 69, 0, 0, 144, 62, 64, 136, 0, 128, 62, 0, 136, 0, 144, 67, 64, 136, 0, 128, 67, 0, 0, 144, 64, 64, 132, 0, 1, 28, 64, 0, 0, 144, 67, 64, 132, 0, 128, 67, 0, 0, 144, 62, 64, 136, 0, 128, 62, 0, 0, 144, 60, 64, 144, 0, 128, 60, 0, 0, 255, 47, 0]]

5.1.2.4. Eliminando o cabeçalho e número de bytes de cada track mTrk

Esta função é desejada para manter nas listas apenas as informações referentes aos eventos MIDI, ou seja, sem o cabeçalho e tamanho de arquivo. Para isto basta eliminar os 8 primeiros Bytes de cada track. Isto é feito da seguinte forma:

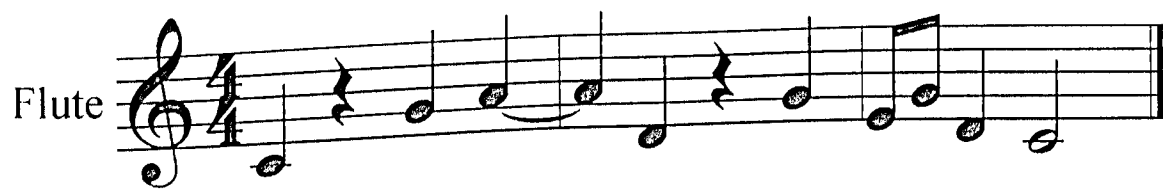


Figura 78

listaTracksMtrksSemCabecalho =: map (drop 8) listaTracksMTrks

A função **map** é uma primitiva do CLEAN que aplica uma determinada função em todas os elementos de uma lista. Como cada elemento da lista é uma lista contendo a informação de um track, apenas a aplicação de map produz o resultado desejado.

O programa

O programa ficou maior do que a janela, portanto não foi possível captura-la. Assim segue-se o programa em modo texto.

```
module listaTracksMtrksSemCabecalho
import StdEnv,StdIO
Start comp
//para abrir o explorer
#    (maybeFile, comp)= selectInputFile comp
#    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
//fim da abertura do explorer
#    (ok, file, comp)= fopen nomeArquivo FReadText comp
#    (conteudo, file)= freads file 300000
#    (ok, comp)= fclose file comp
#listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
#mThd =: take 14 listaTracks
#mTrks =: drop 14 listaTracks
#nTracks =: (mThd!!10)*256 + (mThd!!11)
#listaTracksMTrks =: (reverse (SeparaTracks mTrks [] nTracks))
#listaTracksMtrksSemCabecalho =: map (drop 8) listaTracksMTrks
=listaTracksMtrksSemCabecalho

/*função que pega a lista com o arquivo MIDI e o converte em uma lista com cada
track em uma lista*/
SeparaTracks mTrks mThdMaisMTrks nTracks
    |nTracks == 0 = mThdMaisMTrks
    |otherwise =
     SeparaTracks (drop tamanhoTrack mTrks)( [take (tamanhoTrack) mTrks]++
                  mThdMaisMTrks) (nTracks -1)
where
 tamanhoTrack =:(  ((mTrks!!4)*256*256*256)+((mTrks!!5)*256*256)+
((mTrks!!6)*256) + (mTrks!!7) +8 )
```

Resultado obtido: Uma lista de tracks contendo apenas informações dos eventos MIDI.

[[0, 255, 88, 4, 4, 2, 24, 8, 0, 255, 89, 2, 0, 0, 0, 255, 81, 3, 7, 161, 32, 224, 0, 255, 47, 0], [0, 255, 3, 5, 70, 108, 117, 116, 101, 0, 192, 73, 0, 144, 60, 64, 136, 0, 128, 60, 0, 136, 0, 144, 67, 64, 136, 0, 128, 67, 0, 0, 144, 69, 64, 144, 0, 128, 69, 0, 0, 144, 62, 64, 136, 0, 128, 62, 0, 136, 0, 144, 67, 64, 136, 0, 128, 67, 0, 0, 144, 64, 64, 132, 0, 128, 64, 0, 0, 144, 67, 64, 132, 0, 128, 67, 0, 0, 144, 62, 64, 136, 0, 128, 62, 0, 0, 144, 60, 64, 144, 0, 128, 60, 0, 0, 255, 47, 0]]

5.1.2.5. Separando os eventos dos tracks

O formato 1 utiliza o primeiro track para informar os parâmetros básicos da música, tais como: armadura de clave, tonalidade, metrônomo, e outros mais. Estes parâmetros são armazenados em estruturas chamadas de Meta Eventos. Os demais tracks registram as informações de eventos musicais, tais como: ativação de notas, mudança do instrumento, etc. Assim, será apresentado como separar os eventos do primeiro track (Meta Eventos) e, posteriormente, como genericamente separar os demais tracks (tracks de eventos musicais).

5.1.2.5.1. Primeiro Track:

Partindo da lista de tracks contendo apenas eventos vista na função anterior, a estrutura destas listas sempre seguirá os seguintes passos já vistos no capítulo sobre MIDI. Assim, um evento ou meta evento inicia por um delta time seguido do código de qual evento o segue. No caso do primeiro track, track de meta eventos, um Meta Evento é identificado pelo código hexadecimal FF_{H} ou 255_{10} em decimal. O fato do primeiro track só possuir meta eventos, isto não impede que os demais tracks eventualmente possuam meta eventos - como é o caso do exemplo adotado[37]. Para separar os meta eventos deve-se proceder da seguinte forma: Procura-se um byte com FF_{H} ou 255_{10}. O segundo byte após ele informa quantos bytes o mesmo conterá. A lista seguinte mostra o primeiro track com os meta eventos: **[0, 255, 88, 4, 4, 2, 24, 8, 0, 255, 89, 2, 0, 0, 0, 255, 81, 3, 7, 161, 32, 224, 0, 255, 47, 0]**. A função a ser criada deverá separar os meta eventos, mantendo seus respectivos Delta Times convertidos

[37] Neste exemplo utilizou-se um meta evento (255 03) para informar o instrumento do track, no caso, Flauta.

em ppq. A separação dos Meta Eventos, do exemplo seguido até aqui, fica da seguinte forma: **[[0, 255, 88, 4, 4, 2, 24, 8] , [0, 255, 89, 2, 0, 0] , [0, 255, 81, 3, 7, 161, 32] , [12288, 255, 47, 0]].**

Separar os eventos em listas é similar ao processo de separação do arquivo MIDI em listas de tracks. O maior problema para a nova função é converter Delta Time para ppq. Nos demais tracks, a quantidade de bytes segue a mesma regra. A diferença é que não será procurado apenas os Meta Eventos $FF_H (255_{10})$, mas, também, os eventos de nota e de sistema, tais como ativar e desativar nota.

A nova função, separar eventos por track, necessitará de algumas funções básicas que serão incorporadas a uma biblioteca para utilização por outras funções e outros aplicativos.

5.1.2.5.2. Convertendo Delta Times para ppq

Esta é a primeira função da biblioteca que será montada aos poucos, de acordo com que a necessidade for surgindo. Para converter delta time para ppq é seguida a seguinte regra:

5.1.2.5.2.1. Primeiro byte do Delta Time menor que 80_H (128_{10}): isto indica que ele possui apenas um byte e que o valor convertido para ppq é o próprio delta time

O programa

deltaTime1BytePpq.icl - D:\ANDRE\Tese\Programas\deltaTime1BytePpq.icl

```
1 module deltaTime1BytePpq
2 import StdEnv
3
4
5 deltaTime :: [Int]-> Int
6 deltaTime listaDeEventos
7
8 |primeiroByte < 128 = primeiroByte
9 |otherwise = abort "Este delta time possui mais de um byte"
10 where
11     primeiroByte =: listaDeEventos!!0
12
13 Start= deltaTime [64,255,3,45,2,1]
```

Figura 79

Resultado obtido:
64

5.1.2.5.2.2. **Primeiro byte do delta time maior que 7F (127_{10}) e o segundo menor ou igual a 7F (127_{10})**: a ppq será calculada conforme texto apresentado no capítulo de MIDI.

O programa

O programa ficou maior do que a janela, portanto não foi possível captura-la. Assim segue-se o programa em modo texto.

```
module deltaTime2BytePpq
import StdEnv

deltaTime :: [Int]-> Int
deltaTime listaDeEventos
//          DELTA TIME COM UM BYTE
|primeiroByte < 128 = primeiroByte

//          DELTA TIME COM DOIS BYTES
|( primeiroByte > 127 &&
   segundoByte  < 128  ) =
   (primeiroByte bitand 64)/64 *2^13+
   (primeiroByte bitand 32)/32 *2^12+
   (primeiroByte bitand 16)/16 *2^11+
   (primeiroByte bitand 8)/8   *2^10+
   (primeiroByte bitand 4)/4   *2^9 +
   (primeiroByte bitand 2)/2   *2^8 +
   (primeiroByte bitand 1)/1   *2^7 +

   (segundoByte bitand 64)/64 *2^6 +
   (segundoByte bitand 32)/32 *2^5 +
   (segundoByte bitand 16)/16 *2^4 +
   (segundoByte bitand 8)/8 *2^3 +
   (segundoByte bitand 4)/4*2^2 +
   (segundoByte bitand 2)/2*2^1 +
   (segundoByte bitand 1)/1*2^0
|otherwise = abort "Este delta time possui mais de dois bytes"
where
     primeiroByte =: listaDeEventos!!0
     segundoByte  =: listaDeEventos!!1
Start= deltaTime [224,0,255,47,0]
```

Resposta:

12288

A função **bitand** [24] utilizada é uma primitiva do CLEAN que faz a operação lógica E (and) entre todos os bits do byte com o valor fornecido. Assim, **(primeiroByte bitand 64)/64 *2^13**, significa o seguinte: Pega-se o primeiro byte = 224_{10} = 11100000_2 e realiza-se a operação bitand com 64_{10} = 01000000_2. O valor a ser operado, no caso o 64, é denominado tecnicamente de máscara [24]. O mesmo serve para verificar a existência de um determinado valor de bit em um byte. No presente caso, se no primeiro byte fornecido na lista de eventos existir um bit um no mesmo lugar da máscara, o sistema devolverá o valor da máscara, caso contrário, devolverá 0 (zero).

Veja, a seguir, como funciona bitand neste caso:

Operação: primeiroByte bitand mascara

	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	**decimal**
	128	64	32	16	8	4	2	1	
máscara	0	1	0	0	0	0	0	0	**64**
primeiroByte	1	1	1	0	0	0	0	0	**224**
Bitand	0	1	0	0	0	0	0	0	**64**

Resposta: **bitand = mascara = 64**

A operação **(primeiroByte bitand 64)/64** faz com que o resultado do bitand seja 1 se o bit mascarado for 1 e 0 se for 0. Finalmente, **(primeiroByte bitand 64)/64 *2^13**, converte o bit localizado para sua ordem de grandeza conforme posição que ocupa no byte que formará a ppq[38].

Os demais cálculos para a ppq seguem o mesmo raciocínio.

O programa, a seguir, mostra a versão completa para Delta Time de 4 bytes, já que nada mais de novo aparece nesta função. O programa ficou maior do que a janela, portanto não foi possível captura-la. Assim segue-se o programa em modo texto.

[38] Vale lembrar que o último bit de cada byte não é contado na conversão para ppq, conforme capítulo de MIDI

```
module deltaTimeBytePpq
import StdEnv

deltaTime :: [Int]-> Int
deltaTime listaDeEventos

//          Delta Time com um byte
|primeiroByte < 128 = primeiroByte

//          Delta Time com dois bytes
//O primeiro Byte é o mais significativo
|(  primeiroByte > 127 &&
    segundoByte  < 128  ) =
    (primeiroByte bitand 64)/64 *2^13+
    (primeiroByte bitand 32)/32 *2^12+
    (primeiroByte bitand 16)/16 *2^11+
    (primeiroByte bitand 8)/8   *2^10+
    (primeiroByte bitand 4)/4   *2^9 +
    (primeiroByte bitand 2)/2   *2^8 +
    (primeiroByte bitand 1)/1   *2^7 +

//O segundo Byte é o menos significativo
    (segundoByte bitand 64)/64 *2^6 +
    (segundoByte bitand 32)/32 *2^5 +
    (segundoByte bitand 16)/16 *2^4 +
    (segundoByte bitand 8)/8 *2^3 +
    (segundoByte bitand 4)/4*2^2 +
    (segundoByte bitand 2)/2*2^1 +
    (segundoByte bitand 1)/1*2^0

//      Delta Time com três bytes
|(  primeiroByte > 127 &&
    segundoByte  > 127 &&
    terceiroByte < 128 ) =

    (primeiroByte bitand 64)/64 *2^20 +
    (primeiroByte bitand 32)/32 *2^19 +
    (primeiroByte bitand 16)/16 *2^18 +
    (primeiroByte bitand 8)/8 *2^17 +
    (primeiroByte bitand 4)/4*2^16 +
    (primeiroByte bitand 2)/2*2^15 +
    (primeiroByte bitand 1)/1*2^14 +

    (segundoByte bitand 64)/64 *2^13 +
    (segundoByte bitand 32)/32 *2^12 +
```

```
   (segundoByte bitand 16)/16 *2^11 +
   (segundoByte bitand 8)/8 *2^10 +
   (segundoByte bitand 4)/4*2^9  +
   (segundoByte bitand 2)/2*2^8  +
   (segundoByte bitand 1)/1*2^7  +

   (terceiroByte bitand 64)/64 *2^6 +
   (terceiroByte bitand 32)/32 *2^5 +
   (terceiroByte bitand 16)/16 *2^4 +
   (terceiroByte bitand 8)/8 *2^3 +
   (terceiroByte bitand 4)/4*2^2 +
   (terceiroByte bitand 2)/2*2^1 +
   (terceiroByte bitand 1)/1*2^0

//          Delta Time com quatro bytes
|(  primeiroByte > 127 &&
   segundoByte  > 127 &&
   terceiroByte > 127 &&
   quartoByte <128 ) =

   (primeiroByte bitand 64)/64 *2^27 +
   (primeiroByte bitand 32)/32 *2^26 +
   (primeiroByte bitand 16)/16 *2^25 +
   (primeiroByte bitand 8)/8 *2^24 +
   (primeiroByte bitand 4)/4*2^23 +
   (primeiroByte bitand 2)/2*2^22 +
   (primeiroByte bitand 1)/1*2^21 +

   (segundoByte bitand 64)/64 *2^20 +
   (segundoByte bitand 32)/32 *2^19 +
   (segundoByte bitand 16)/16 *2^18 +
   (segundoByte bitand 8)/8 *2^17 +
   (segundoByte bitand 4)/4*2^16+
   (segundoByte bitand 2)/2*2^15 +
   (segundoByte bitand 1)/1*2^14 +

   (terceiroByte bitand 64)/64 *2^13 +
   (terceiroByte bitand 32)/32 *2^12 +
   (terceiroByte bitand 16)/16 *2^11 +
   (terceiroByte bitand 8)/8 *2^10 +
   (terceiroByte bitand 4)/4*2^9+
   (terceiroByte bitand 2)/2*2^8 +
   (terceiroByte bitand 1)/1*2^7 +

   (quartoByte bitand 64)/64 *2^6 +
```

```
   (quartoByte bitand 32)/32 *2^5 +
   (quartoByte bitand 16)/16 *2^4 +
   (quartoByte bitand 8)/8 *2^3 +
   (quartoByte bitand 4)/4*2^2+
   (quartoByte bitand 2)/2*2^1 +
   (quartoByte bitand 1)/1*2^0
//         Mensagem de erro para Delta Time > 4 bytes
|otherwise = abort "Nao existe deltaTime com mais de quarto bytes!"
//         Constantes utilizadas
where
     primeiroByte =: listaDeEventos!!0
     segundoByte  =: listaDeEventos!!1
     terceiroByte =: listaDeEventos!!2
     quartoByte   =: listaDeEventos!!3

Start= deltaTime [224,180,257,97,87]
```

Resultado obtido:

202178785

5.1.2.5.3. Separando os eventos dos demais tracks

Esta função se encarrega de criar uma lista para cada evento, colocando na cabeça da lista o valor do Delta Time convertido para ppq, utilizando, para isto, a função de conversão já depositada na biblioteca: "**MIDIBiblioteca.icl**".

Para separar o evento do restante da lista, deve-se identificar, primeiramente, de qual evento se trata. O número de bytes de cada evento é fixo, menos no caso dos Meta Eventos que trazem em seu corpo a quantidade de bytes que cada um conterá. Se o primeiro byte do delta time for maior que 7F (127_{10}) e o segundo menor ou igual a 7F (127_{10}), isto significa que se deve armazenar o valor convertido do mesmo como primeiro elemento de uma nova lista e os próximos 3 bytes (no caso de 9n e 8n) como demais elementos. Se o primeiro byte do delta time for maior que 7F (127_{10}), o segundo byte do delta time for maior que 7F (127_{10}) e o terceiro for menor ou igual a 7F (127_{10}), isto significa que se deve armazenar o valor convertido do mesmo como primeiro elemento de uma nova lista e os próximos 3 bytes (no caso de 9n e 8n) como demais elementos. Se o primeiro byte do delta time for maior que 7F (127_{10}), o segundo byte do delta time for maior que 7F (127_{10}), o terceiro byte do delta time for

maior que 7F (127_{10}) e o quarto byte for menor ou igual a 7F (127_{10}), isto significa que deve-se armazenar o valor convertido do mesmo como primeiro elemento de uma nova lista e os próximos 3 bytes (no caso de 9n e 8n) como demais elementos.

Como na totalidade dos casos atuais os Delta Times possuem até dois bytes , o que equivale a uma unidade de tempo para uma nota com tempo superior a 32 semínimas[39], foi implementado o algoritmo para os tracks de notas para delta time com até dois bytes. Apenas para o primeiro track, o dos metaeventos, foi implementado o algoritmo de conversão dos Delta Times dos metaeventos para quatro bytes, para atender a resolução do programa Finale[40].

O programa

O programa ficou maior do que a janela, portanto não foi possível captura-la. Assim segue-se o programa em modo texto.

```
module listaEventosPorTrack
import StdEnv,StdIO,MIDIBiblioteca

Start comp
//para abrir o explorer
#    (maybeFile, comp)= selectInputFile comp
#    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
//fim da abertura do explorer
#    (ok, file, comp)= fopen nomeArquivo FReadText comp
#    (conteudo, file)= freads file 300000
#    (ok, comp)= fclose file comp
#listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
#mThd =: take 14 listaTracks
#mTrks =: drop 14 listaTracks
#nTracks =: (mThd!!10)*256 + (mThd!!11)
#listaTracksMTrks =: (reverse (SeparaTracks mTrks [] nTracks))
#listaTracksMtrksSemCabecalho =: map (drop 8) listaTracksMTrks
#listaEventosPorTrack =: map separaTrack listaTracksMtrksSemCabecalho
=listaEventosPorTrack

/* função que pega a lista com o arquivo MIDI e o converte em uma lista com cada
```

[39] valor este que não possui significado prático ou musical

[40] No finale, o delta time do último metaevento, no track 1, formato 1, permite o cálculo da duração da música, juntamente com o valor do metrônomo.

```
track em uma lista */

SeparaTracks mTrks mThdMaisMTrks nTracks
    |nTracks == 0 = mThdMaisMTrks
    |otherwise =
     SeparaTracks (drop tamanhoTrack mTrks)( [take (tamanhoTrack) mTrks]++
                                                mThdMaisMTrks) (nTracks -1)

where
 tamanhoTrack =:(  ((mTrks!!4)*256*256*256)+((mTrks!!5)*256*256)+
                                    ((mTrks!!6)*256) + (mTrks!!7) +8 )

/*
A função  lerEvento a seguir, se encarrega de determinar o número de bytes do  Delta
Time, convertê-lo para ppq e identificar o evento, Armazenando-o em uma lista cuja
cabeça é o número de bytes desta lista. Ela usa as funções listaMetaEvento.
Exemplo:
Start = lerEvento [129,2,255,47,4,1,2,3,4,126,143,60 ,100]
devolve [9,130,255,47,4,1,2,3,4], onde 9 é a quantidade de elementos
desta nova lista e 130 é o deltaTime convertido em ppq (129 e 2 = 129 -128 = 1.
1 = 128 (desloca um bit para a direita) =>  128+2 = 130)
Se tiver 3 bytes, vamos ver como fica:

Start = lerEvento [129,146,2,255,47,5,1,2,3,4,126,143,60 ,100]
devolve: [10,35074,255,47,5,1,2,3,4,126]
*/
lerEvento listaTracksMtrksSemCabecalho
/*A cláusula seguinte testa se o evento é um metaEvento FF = 255, e se o delta time é
de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho == 255) =
            [(length(listaMetaEvento listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!3))):
                (listaMetaEvento listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!3))]

/*A cláusula seguinte testa se o evento é um metaEvento FF = 255, e se o delta time é
de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento2 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!4))+1 ):
/* +1 devido ao contador ser a quantidade de bytes do evento inicial, como tem 2 bytes
de delta time, tem-se uma contagem a mais para três bytes. */
```

```
                (listaMetaEvento2 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!4))]

/*A cláusula seguinte testa se o evento é um metaEvento FF = 255 e se o delta time é
de três bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==3) &&
        (next3 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento3 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!5))+2 ):
                (listaMetaEvento3 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!5))]

/*A cláusula seguinte testa se o evento é um metaEvento FF = 255 e se o delta time é
de quatro bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==4) &&
        (next4 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento4 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!6))+3 ):
                (listaMetaEvento4 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!6))

/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144, 9F =159.
E testa se o delta time é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 144)&&
        (next listaTracksMtrksSemCabecalho <= 159)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144, 9F =159.
E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 144)&&
        (next2 listaTracksMtrksSemCabecalho <= 159)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]
/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128, 8F =
143. E testa se o delta time é de um byte */
    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
```

```
        (next listaTracksMtrksSemCabecalho >= 128)&&
        (next listaTracksMtrksSemCabecalho <= 143)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128, 8F =
143. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 128)&&
        (next2 listaTracksMtrksSemCabecalho <= 143)=
            [(length[(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
                (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 192)&&
        (next listaTracksMtrksSemCabecalho <= 207)=
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 192)&&
    (next2 listaTracksMtrksSemCabecalho <= 207)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 176)&&
        (next listaTracksMtrksSemCabecalho <= 191)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]
```

```
/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 176)&&
        (next2 listaTracksMtrksSemCabecalho <= 191)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

    | otherwise= []

//mensagem de pressão na tecla
/*A cláusula seguinte testa se o evento é um evento de mensagem de pressão na tecla,
e testa se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 160)&&
        (next listaTracksMtrksSemCabecalho <= 175)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de pressão na tecla
Exemplo: B7 = VolumeE testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 160)&&
        (next2 listaTracksMtrksSemCabecalho <= 175)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

/* Para os outros eventos basta seguir os passos dados */

    | otherwise= []

//fim mensagem pressão na tecla 2 bytes
//mensagem de variação do pitch bend 2 bytes
/*A cláusula seguinte testa se o evento é um evento variação do pitch bend e testa se o
delta time é de um byte. */
```

```
    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 224)&&
        (next listaTracksMtrksSemCabecalho <= 239)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 224)&&
        (next2 listaTracksMtrksSemCabecalho <= 239)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

/* Para os outros eventos basta seguir os passos dados */
    | otherwise= []

//fim do pitch bend
//mensagem pressão no teclado 1 byte

/*A cláusula seguinte testa se o evento é um evento de pressão no teclado e testa se o
delta time é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 208)&&
        (next listaTracksMtrksSemCabecalho <= 223)=
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time
é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 208)&&
    (next2 listaTracksMtrksSemCabecalho <= 223)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]
```

```
//fim de pressão no teclado

// xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
//rotina para separar um track em uma lista com listas de eventos

separaTrack trackLista = tl(reverse (SeparaEvento trackLista [[]]))
SeparaEvento trackLista listaFinal
     |(length trackLista) == 0 = listaFinal
     |otherwise = SeparaEvento
                    (drop (hd evento) trackLista)
                    [(tl evento)]++listaFinal
where
 evento =: lerEvento trackLista

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
/* recebe uma lista, pega lista de metaevento, converte o Delta Time para ppq e
devolve a lista com o deltaTimePpq na cabeça e o metaEvento completo. */

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
listaMetaEvento listaTracksMtrksSemCabecalho tamanho  = take (tamanho + 4)
listaTracksMtrksSemCabecalho

/* listaMetaEvento devolve o metaEvento com cabeçalho e delta time (para delta time
= 1 byte) */
listaMetaEvento2 listaTracksMtrksSemCabecalho tamanho =
     [(deltaTime listaTracksMtrksSemCabecalho):
      (take (tamanho +3) (drop 2 listaTracksMtrksSemCabecalho))]

//listaMetaEvento2 = idem listaMetaEvento para deltaTime = 2 bytes

listaMetaEvento3  listaTracksMtrksSemCabecalho  tamanho  =
   [(deltaTime listaTracksMtrksSemCabecalho):
    (take (tamanho +3) (drop 3 listaTracksMtrksSemCabecalho))]

//listaMetaEvento3 = idem listaMetaEvento2 para deltaTime = 3 bytes

listaMetaEvento4  listaTracksMtrksSemCabecalho  tamanho  =
   [(deltaTime listaTracksMtrksSemCabecalho):
    (take (tamanho +3) (drop 4 listaTracksMtrksSemCabecalho))]

//listaMetaEvento4 = idem listaMetaEvento2 para deltaTime = 3 bytes
```

Resultado obtido: Lista de eventos por track

[[[0, 255, 88, 4, 4, 2, 24, 8] , [0, 255, 89, 2, 0, 0] , [0, 255, 81, 3, 7, 161, 32] , [12288, 255, 47, 0]] , [[0, 255, 3, 5, 70, 108, 117, 116, 101] , [0, 192, 73] , [0, 144, 60, 64] , [1024, 128, 60, 0] , [1024, 144, 67, 64], [1024, 128, 67, 0], [0, 144, 69, 64], [2048, 128, 69, 0], [0, 144, 62, 64], [1024, 128, 62, 0] , [1024, 144, 67, 64], [1024, 128, 67, 0], [0, 144, 64, 64], [512, 128, 64, 0], [0, 144, 67, 64], [512, 128, 67, 0], [0, 144, 62, 64], [1024, 128, 62, 0], [0, 144, 60, 64], [2048, 128, 60, 0], [0, 255, 47, 0]]]

A partitura do exemplo pode levar a uma conclusão errada sobre o método de determinação da duração de cada nota. Observe:

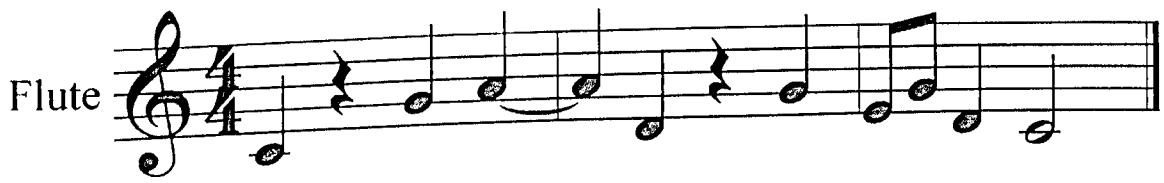


Figura 80

Veja a representação desta partitura no segundo track da lista de eventos por track:
[[0, 255, 3, 5, 70, 108, 117, 116, 101] , [0, 192, 73] , [0, 144, 60, 64] , [1024, 128, 60, 0] , [1024, 144, 67, 64] , [1024, 128, 67, 0] , [0, 144, 69, 64] , [2048, 128, 69, 0] , [0, 144, 62, 64] , [1024, 128, 62, 0] , [1024, 144, 67, 64] , [1024, 128, 67, 0] , [0, 144, 64, 64] , [512, 128, 64, 0] , [0, 144, 67, 64] , [512, 128, 67, 0] , [0, 144, 62, 64] , [1024, 28, 62, 0] , [0, 144, 60, 64] , [2048, 128, 60, 0] , [0, 255, 47, 0]]

A seguir é feita uma análise deste track:

[0,255,3,5,70,108,117,116,101] = Esta linha indica através do Meta Evento (255, 3) que o instrumento escolhido para tocar é Flute: 70 = F, 108 = l, 117 = u, 116 = t e 101 = e.

[0,192,73] = Evento mudança de instrumento (192) para Flute (73).

[0,144,60,64] = Ativar (144) nota Dó (60) com volume 64.

[1024,128,60,0] = Espera-se o tempo de uma semínima (ppq=1024) e desativa (128) a nota Dó (60).

[1024,144,67,64]	= Espera-se o tempo de uma semínima e ativa a nota Sol (67) com volume 64.
[1024,128,67,0]	= Espera-se o tempo de uma semínima e desativa a nota Sol
[0,144,69,64]	= Ativa-se imediatamente (delta time = 0) a nota Lá (69) com volume 64.
[2048,128,69,0]	= Espera-se o tempo de duas semínimas(2048) e desativa-se a nota Lá.
[0,144,62,64]	= Ativa-se imediatamente a nota Ré com volume 64.
[1024,128,62,0]	= .....
[1024,144,67,64]	= .....
[1024,128,67,0]	= .....
[0,144,64,64]	= .....
[512,128,64,0]	= .....
[0,144,67,64]	= .....
[512,128,67,0]	= .....
[0,144,62,64]	= .....
[1024,128,62,0]	= .....
[0,144,60,64]	= .....
[2048,128,60,0]	= .....
[0,255,47,0]	= Fim de track.

Ao se analisar o exemplo, nota-se que logo após a ativação da nota o delta time de desativação seguinte indica a figura musical da nota anterior. Isto ocorre devido antes de iniciar uma nova nota, desativar-se a anterior. O que ocorre, musicalmente, é que se pode ativar uma nota e deixa-la soando enquanto outras notas são ativadas e desativadas. Nestes casos, fica bem mais complexo determinar a figura musical de cada nota. O exemplo seguinte ilustra este problema:

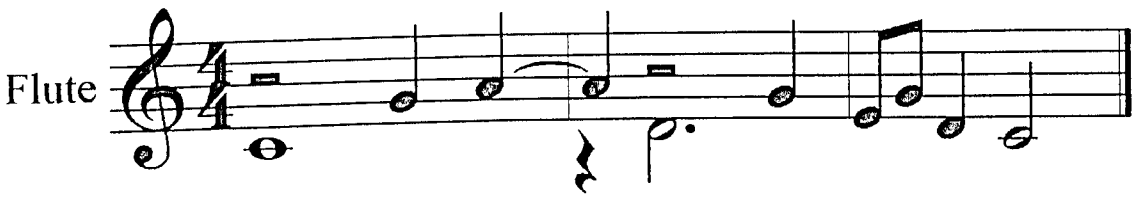


Figura 81

A lista do segundo track com as informações da melodia fica:

[[0, 255, 3, 5, 70, 108, 117, 116, 101], [0, 192, 73], [0, 144, 60, 64], [2048, 144, 67, 64], [1024, 128, 67, 0], [0, 144, 69, 64], [1024, 128, 60, 0], [1024, 128, 69, 0], [0, 144, 62, 64], [2048, 144, 67, 64], [1024, 128, 67, 0], [0, 128, 62, 0], [0, 144, 64, 64], [512, 128, 64, 0], [0, 144, 67, 64], [512, 128, 67, 0], [0, 144, 62, 64], [1024, 128, 62, 0], [0, 144, 60, 64], [2048, 128, 60, 0], [0, 255, 47, 0]]

A primeira nota ativada, a nota dó, durará todo o primeiro compasso. Durante este tempo, a nota sol será ativada e desativada e a nota lá será ativada. Apenas depois de decorrido este tempo é que a nota dó será desativada. Desta forma, o próximo delta time após ativação da nota não é o valor da duração da mesma. Neste caso, o tempo de duração da nota deverá ser computado através da soma de todos os Delta Times existentes entre a ativação e a desativação (inclusive) desta mesma nota. Na lista a seguir, ilustra-se em **negrito** o início e o término da nota **dó** e, em *itálico*, o início e o término da nota *ré*.

[0,255,3,5,70,108,117,116,101] = Esta linha indica através do Meta Evento (255, 3) que o instrumento escolhido para tocar é Flute: 70 = F, 108 = l, 117 = u, 116 = t e 101 = e.

[0,192,73] = Evento mudança de instrumento (192) para Flute (73).

[0,144,60,64] = **Ativar (144) nota Dó (60) com volume 64.**

[2048,144,67,64] = ...

[1024,128,67,0] = ...

[0,144,69,64]	= .....
[1024,128,60,0]	= **Desativar (128) a nota Dó (60).**
[1024,128,69,0]	= .....
[0,144,62,64]	= *Ativação (144) da nota Ré (62).*
[2048,144,67,64]	= .....
[1024,128,67,0]	= .....
[0,128,62,0]	= *Desativação (128) da nota Ré (62).*
[0,144,64,64]	= .....
[512,128,64,0]	= .....
[0,144,67,64]	= .....
[512,128,67,0]	= .....
[0,144,62,64]	= .....
[1024,128,62,0]	= .....
[0,144,60,64]	= .....
[2048,128,60,0]	= .....
[0,255,47,0]	= Fim de track.

a) Pode-se perceber que o valor, a duração da nota Dó será a totalização dos Delta Times após ela até sua desativação, ou seja: 2048 + 1024 + 0+ 1024 = 4096 = 4 semínimas (1024) ou uma semibreve.

b) A duração da nota Ré é totalizada por: 2048 + 1024 + 0 = 3072 = uma mínima mais uma semínima, ou uma mínima pontuada.

Observado os exemplos, pode-se determinar um algoritmo que gere uma nova lista de tracks contendo em cada lista de evento, a duração do evento na cabeça da lista e na calda o evento. Para isto, deve-se determinar o tempo de cada evento e eliminar das listas de tracks os eventos de desativação de notas, assim, este novo formato de apresentação será uma cópia mais fiel da partitura grafada.

5.1.2.6. Conversão de ppq em Delta Time

Para implementar esta função, necessita-se apenas conhecer o valor da ppq. Esta função será utilizada como uma biblioteca para outras funções mais complexas.

O programa

ppqParaDeltaT.icl - D:\ANDRE\Tese\Programas\ppqParaDeltaT.icl

```
module ppqParaDeltaT
import StdEnv

/*função que transforma ppq em uma lista de
bytes de deltaTimes*/

ppqToDeltaTime :: Int -> [Int]
ppqToDeltaTime x
|x<=127 = [x]
|otherwise = ppqToDT (x/128)   [(x rem 128)]
ppqToDT x valorDT
 |x <= 127 =   [(x+128):valorDT]
 |x >= 128 =   ppqToDT ( x/128 ) [((x rem 128)+128):valorDT]

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Start = ppqToDeltaTime 128
```

Figura 82

Resultado obtido:

[129,0]

Ou seja, o delta-time resultante possui dois bytes. O primeiro é 129 (81_H) e o segundo 0 (00_H).

5.1.2.7. Metrônomo

Esta função inicialmente pode parecer simples de ser implementada, contudo este é um engano natural. A complexidade é que não existe uma informação no arquivo MIDI que forneça o valor do metrônomo diretamente. A informação existente é o tempo de uma semínima em microssegundos. Esta informação é disponibilizada

em um meta-evento denominado Set-Tempo[41]. Assim, deve-se localizar este Meta Evento e transformar o tempo do mesmo em batidas (semínimas) por minuto.

A função final que calcula o valor do metrônomo é:

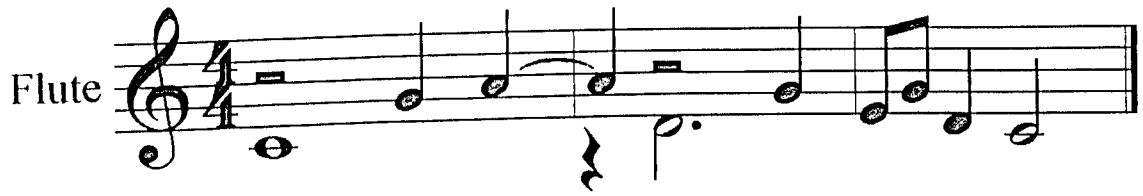


Figura 83

Metrônomo = 60 / (tempo de uma semínima)

A função em CLEAN será:

```
#metronomo =: [60000000/(((x!!4)*256*256)+((x!!5)*256)+(x!!6))
                \\x<-TrackDeMetaEventos
                  | (x!!1 == 255) && (x!!2 ==81)]
```

Ou seja, como o tempo da semínima é dado em microssegundos, tem-se que o valor do metrônomo será **60.000.000** dividido pelo tempo desta semínima. O Meta Evento tempo é o **81**. Localizado o Meta Evento, o tempo da semínima, em microssegundos, está registrados nos bytes **5, 6 e 7**. A função Zermelo-Frankel diz ao CLEAN procurar no track de meta-eventos uma lista que comece pelo byte cujo valor é **255** seguido do valor **81**, ou seja: **x<-TrackDeMetaEventos | (x!!1 == 255) && (x!!2 ==81)**. Ao encontá-la, o tempo é calculado como indicado: **60000000 / (((x!!4) * 256 * 256) + ((x!!5) * 256) + (x!!6))**. O restante do código é devido a separação dos eventos de cada track para posterior identificação.

O programa:

```
Module metronomo
import StdEnv, StdIO

Start comp
//para abrir o explorer
```

[41] Ver capítulo sobre MIDI.

```
#     (maybeFile, comp) = selectInputFile comp
#     (Just nomeArquivo)= maybeFile //Retira nome de arquivo
#arquivoLido = nomeArquivo
//fim da abertura do explorer

#     (ok, file, comp)= fopen nomeArquivo FReadText comp
#     (conteudo, file)= freads file 300000

#listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
#ppq =: ((listaTracks !! 12)*256) + (listaTracks !! 13)
#mThd =: take 14 listaTracks
#mTrks =: drop 14 listaTracks
#nTracks =: (mThd!!10)*256 + (mThd!!11)
#listaTracksMTrks =: (tl (reverse (SeparaTracks mTrks [mThd] nTracks)))
#listaTracksMtrksSemCabecalho =: map (drop 8) listaTracksMTrks
#listaEventosPorTrack =: map separaTrack listaTracksMtrksSemCabecalho
#TrackDeMetaEventos =: listaEventosPorTrack !! 0
#metronomo =: [60000000/(((x!!4)*256*256)+((x!!5)*256)+(x!!6))
              \\x<-TrackDeMetaEventos
                    | (x!!1 == 255) && (x!!2 ==81)]

= metronomo

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* função que pega a lista com o arquivo MIDI e o converte //em uma lista com cada track em
uma lista */

SeparaTracks mTrks mThdMaisMTrks nTracks
     |nTracks == 0 = mThdMaisMTrks
     |otherwise =
      SeparaTracks (drop tamanhoTrack mTrks) ( [take (tamanhoTrack) mTrks]++
                                               mThdMaisMTrks) (nTracks -1)
where
 tamanhoTrack =:(  ((mTrks!!5)*256*256)+ ((mTrks!!6)*256) + (mTrks!!7) +8 )

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Rotina para separar um track em uma lista com listas de eventos

separaTrack trackLista = tl(reverse (SeparaEvento trackLista [[]]))
SeparaEvento trackLista listaFinal
     |(length trackLista) == 0 = listaFinal
     |otherwise = SeparaEvento
                  (drop (hd evento) trackLista)
                  |(tl evento)]++listaFinal
where
  evento =: lerEvento trackLista
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
/* A função  lerevento a seguir, se encarrega de determinar o número de bytes do  deltatime,
convertê-lo para ppq e identificar o evento, armazenando-o em uma lista cuja cabeça é o
número de bytes desta lista.Ela usa as funções listaMetaEvento...
Exemplo:
Start = lerEvento [129,2,255,47,4,1,2,3,4,126,143,60 ,100]
devolve [9,130,255,47,4,1,2,3,4], onde 9 é a quantidade de elementos
desta nova lista e 130 é o deltaTime convertidoem ppq (129 e 2 = 129 -128 = 1.
1 = 128(desloca um bit.  para a direita)=>  128+2 = 130). Se tiver 3 bytes, fica:
Start = lerEvento [129,146,2,255,47,5,1,2,3,4,126,143,60 ,100]
devolve: [10,35074,255,47,5,1,2,3,4,126]  */
lerEvento listaTracksMtrksSemCabecalho

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é de um
byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho == 255) =
            [(length(listaMetaEvento listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!3))):
                (listaMetaEvento listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!3))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é de dois
bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento2 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!4))+1 ):

/* +1 devido ao contador ser a quantidade de bytes do evento inicial, como tem 2 bytes de
delta time, tem-se uma contagem a mais para três bytes duas contagens a mais, ... */

                (listaMetaEvento2 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!4))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é de três
bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==3) &&
        (next3 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento3 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!5))+2 ):
                (listaMetaEvento3 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!5))]
/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é de quatro
bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==4) &&
```

```
        (next4 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento4 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!6))+3 ):
                (listaMetaEvento4 listaTracksMtrksSemCabecalho
                (listaTracksMtrksSemCabecalho!!6))]

/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144, 9F=159. E testa
se o delta time é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 144)&&
        (next listaTracksMtrksSemCabecalho <= 159)=
            [length (take 4 listaTracksMtrksSemCabecalho):
            (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144, 9F=159. E testa
se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 144)&&
        (next2 listaTracksMtrksSemCabecalho <= 159)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128, 8F = 143. E
testa se o delta time é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 128)&&
        (next listaTracksMtrksSemCabecalho <= 143)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128, 8F = 143. E
testa se o delta time é de dois bytes    */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 128)&&
        (next2 listaTracksMtrksSemCabecalho <= 143)=
            [(length[(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
                (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]
/*A cláusula seguinte testa se o evento é um evento de program change (mudar o instrumento)
C0 = 192, CF = 207. E testa se o delta time de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
```

```
        (next listaTracksMtrksSemCabecalho >= 192)&&
        (next listaTracksMtrksSemCabecalho <= 207)=
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o instrumento)
C0 = 192, CF = 207. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 192)&&
    (next2 listaTracksMtrksSemCabecalho <= 207)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos, sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 176)&&
        (next listaTracksMtrksSemCabecalho <= 191)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos, sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 176)&&
        (next2 listaTracksMtrksSemCabecalho <= 191)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]
    | otherwise= []

//mensagem de pressão na tecla
/*A cláusula seguinte testa se o evento é um evento de mensagem de pressão na tecla e testa
se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 160)&&
        (next listaTracksMtrksSemCabecalho <= 175)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de pressão na tecla
```

```
Exemplo: B7 = VolumeE testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 160)&&
        (next2 listaTracksMtrksSemCabecalho <= 175)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

    | otherwise= []

//fim mensagem pressão na tecla 2 bytes

//mensagem de variação do pitch bend 2 bytes

/*A cláusula seguinte testa se o evento é um evento  variação do pitch bend e testa se o delta
time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 224)&&
        (next listaTracksMtrksSemCabecalho <= 239)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos, sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 224)&&
        (next2 listaTracksMtrksSemCabecalho <= 239)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]
// Para os outros evenvtos deve-se seguir os mesmos passos.

    | otherwise= []

//fim do pitch bend

//mensagem pressão no teclado 1 byte
/*A cláusula seguinte testa se o evento é um evento de pressão no teclado. E testa se o delta
time de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 208)&&
        (next listaTracksMtrksSemCabecalho <= 223)=
```

```
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o instrumento)
C0 = 192, CF = 207. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 208)&&
    (next2 listaTracksMtrksSemCabecalho <= 223)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]

//fim de pressão no teclado

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Recebe uma lista, pega lista de metaevento, converte o Delta Time para ppq e devolve a
lista com o deltaTimePpq na cabeça e o metaEvento completo. */

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

listaMetaEvento listaTracksMtrksSemCabecalho tamanho  = take
            (tamanho + 4) listaTracksMtrksSemCabecalho

/* listaMetaEvento devolve o metaEvento com cabeçalho e delta time (para delta time = 1
byte) */

listaMetaEvento2 listaTracksMtrksSemCabecalho tamanho =
            [(deltaTime listaTracksMtrksSemCabecalho):
                        (take (tamanho +3) (drop 2 listaTracksMtrksSemCabecalho))]

//listaMetaEvento2 = idem listaMetaEvento para deltaTime = 2 bytes

listaMetaEvento3  listaTracksMtrksSemCabecalho  tamanho  =
    [(deltaTime listaTracksMtrksSemCabecalho):
    (take (tamanho +3) (drop 3 listaTracksMtrksSemCabecalho))]

//listaMetaEvento3 = idem listaMetaEvento2 para deltaTime = 3 bytes

listaMetaEvento4  listaTracksMtrksSemCabecalho  tamanho  =
  [(deltaTime listaTracksMtrksSemCabecalho):
   (take (tamanho +3) (drop 4 listaTracksMtrksSemCabecalho))]
//listaMetaEvento4 = idem listaMetaEvento2 para deltaTime = 3 bytes

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

NBytesDeltaTime :: [Int] -> Int
NBytesDeltaTime lista
```

```
| (primeiroByte < 128)  = 1 // UM BYTE
|( (primeiroByte > 127) &&
  (segundoByte  < 128) )= 2 // DOIS BYTES
|( (primeiroByte > 127) &&
  (segundoByte  > 127) &&
  (terceiroByte < 128) )= 3  // TRÊS BYTES
|otherwise = 4 // quatro bytes
where
 primeiroByte =: lista!!0
 segundoByte  =: lista!!1
 terceiroByte =: lista!!2

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Pega elementos de uma lista

next :: [Int]-> Int
next lista  = lista!!1

next2 :: [Int]-> Int
next2 lista = lista!!2

next3 :: [Int]-> Int
next3 lista = lista!!3
next4 :: [Int]-> Int
next4 lista = lista!!4

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Esta função converte uma lista de bytes de deltaTime para ppq

deltaTime :: [Int]-> Int
deltaTime novaListaTracksMtrksSemCabecalho

//          Delta Time com 1 byte
|primeiroByte < 128 = primeiroByte // deltaTime4 com 1 byte
//          Delta Time com dois bytes
|(  primeiroByte > 127 &&
    segundoByte  < 128  ) =
    (primeiroByte bitand 64)/64 *2^13+
    (primeiroByte bitand 32)/32 *2^12+
    (primeiroByte bitand 16)/16 *2^11+
    (primeiroByte bitand 8)/8   *2^10+
    (primeiroByte bitand 4)/4   *2^9 +
    (primeiroByte bitand 2)/2   *2^8 +
    (primeiroByte bitand 1)/1   *2^7 +

    (segundoByte bitand 64)/64 *2^6 +
    (segundoByte bitand 32)/32 *2^5 +
```

```
   (segundoByte bitand 16)/16 *2^4 +
   (segundoByte bitand 8)/8 *2^3 +
   (segundoByte bitand 4)/4*2^2 +
   (segundoByte bitand 2)/2*2^1 +
   (segundoByte bitand 1)/1*2^0

//       Delta Time com três bytes
|(  primeiroByte > 127 &&
   segundoByte  > 127 &&
   terceiroByte < 128 ) =

   (primeiroByte bitand 64)/64 *2^20 +
   (primeiroByte bitand 32)/32 *2^19 +
   (primeiroByte bitand 16)/16 *2^18 +
   (primeiroByte bitand 8)/8 *2^17 +
   (primeiroByte bitand 4)/4*2^16 +
   (primeiroByte bitand 2)/2*2^15 +
   (primeiroByte bitand 1)/1*2^14 +

   (segundoByte bitand 64)/64 *2^13 +
   (segundoByte bitand 32)/32 *2^12 +
   (segundoByte bitand 16)/16 *2^11 +
   (segundoByte bitand 8)/8 *2^10 +
   (segundoByte bitand 4)/4*2^9  +
   (segundoByte bitand 2)/2*2^8  +
   (segundoByte bitand 1)/1*2^7  +

   (terceiroByte bitand 64)/64 *2^6 +
   (terceiroByte bitand 32)/32 *2^5 +
   (terceiroByte bitand 16)/16 *2^4 +
   (terceiroByte bitand 8)/8 *2^3 +
   (terceiroByte bitand 4)/4*2^2 +
   (terceiroByte bitand 2)/2*2^1 +
   (terceiroByte bitand 1)/1*2^0

//            Delta Time com quatro bytes
|(  primeiroByte > 127 &&
   segundoByte  > 127 &&
   terceiroByte > 127 &&
   quartoByte <128 ) =

   (primeiroByte bitand 64)/64 *2^27 +
   (primeiroByte bitand 32)/32 *2^26 +
   (primeiroByte bitand 16)/16 *2^25 +
   (primeiroByte bitand 8)/8 *2^24 +
   (primeiroByte bitand 4)/4*2^23 +
   (primeiroByte bitand 2)/2*2^22 +
   (primeiroByte bitand 1)/1*2^21 +
```

```
  (segundoByte bitand 64)/64 *2^20 +
  (segundoByte bitand 32)/32 *2^19 +
  (segundoByte bitand 16)/16 *2^18 +
  (segundoByte bitand 8)/8 *2^17 +
  (segundoByte bitand 4)/4*2^16+
  (segundoByte bitand 2)/2*2^15 +
  (segundoByte bitand 1)/1*2^14 +

  (terceiroByte bitand 64)/64 *2^13 +
  (terceiroByte bitand 32)/32 *2^12 +
  (terceiroByte bitand 16)/16 *2^11 +
  (terceiroByte bitand 8)/8 *2^10 +
  (terceiroByte bitand 4)/4*2^9+
  (terceiroByte bitand 2)/2*2^8 +
  (terceiroByte bitand 1)/1*2^7 +

  (quartoByte bitand 64)/64 *2^6 +
  (quartoByte bitand 32)/32 *2^5 +
  (quartoByte bitand 16)/16 *2^4 +
  (quartoByte bitand 8)/8 *2^3 +
  (quartoByte bitand 4)/4*2^2+
  (quartoByte bitand 2)/2*2^1 +
  (quartoByte bitand 1)/1*2^0

//        Mensagem de erro para deltatime > 4 bytes
|otherwise = abort "Nao existe deltaTime com mais de quarto bytes!"

//        Constantes utilizadas

where
     primeiroByte =: novaListaTracksMtrksSemCabecalho!!0
     segundoByte  =: novaListaTracksMtrksSemCabecalho!!1
     terceiroByte =: novaListaTracksMtrksSemCabecalho!!2
     quartoByte   =: novaListaTracksMtrksSemCabecalho!!3
```

Resultado obtido:

[120]

5.1.2.8. Tonalidade da música

Apesar de existir um Meta Evento que indica a tonalidade da música[42] (Meta Evento 255, 59), é necessário, antes, também separar o arquivo em tracks e os tracks em eventos de traks para identificação da lista com o referido Meta Evento.

[42] Ver capítulo sobre MIDI para maiores detalhes.

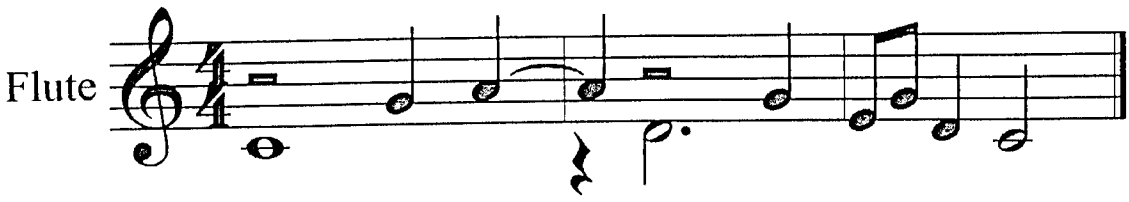


Figura 84

O programa:

```
module tonalidade
import StdEnv, StdIO

Start comp

//para abrir o explorer
#    (maybeFile, comp)= selectInputFile comp
#    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
#arquivoLido = nomeArquivo
//fim da abertura do explorer
#    (ok, file, comp)= fopen nomeArquivo FReadText comp
#    (conteudo, file)= freads file 300000

#listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
#ppq =: ((listaTracks !! 12)*256) + (listaTracks !! 13)
#mThd =: take 14 listaTracks
#mTrks =: drop 14 listaTracks
#nTracks =: (mThd!!10)*256 + (mThd!!11)
#listaTracksMTrks =: (tl (reverse (SeparaTracks mTrks [mThd] nTracks)))
#listaTracksMtrksSemCabecalho =: map (drop 8) listaTracksMTrks
#listaEventosPorTrack =: map separaTrack listaTracksMtrksSemCabecalho
#TrackDeMetaEventos =: listaEventosPorTrack !! 0
#tonalidadeEscrita=:  mostraTonalidade (veSeTem TrackDeMetaEventos)
/* devolve o codigo da tonalidade. Se for bemol, tem casa da dezena no primeiro
elemento da tupla:
Ex. normal -> (1,0) um sustenido e tom maior
Ex. bemol -> (10,0) um bemol e  tom maior */
#tonalidadeParaCalculo =: calculaAcidentes(hd (veSeTem TrackDeMetaEventos))

=tonalidadeEscrita

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//localiza metaEvento tonalidade (nem todos os software têm

veSeTem trackDeMetaEventos
```

```
/* Devolve [[256,256]] se não tiver meta evento tonalidade, como é o caso do
Encore, se tiver, devolve a lista [[acid, modo]] */

|[x\\x<-trackDeMetaEventos |
      (x!!1 == 255) &&
      (x!!2==89)]== [] = [[256,256]]

// Tonalidade

|[x\\x<-trackDeMetaEventos |
      (x!!1 == 255) &&
      (x!!2==89)]<> [] =
            |[(x!!4), (x!!5)]|\\x<-trackDeMetaEventos |
                  (x!!1 == 255) &&
                  (x!!2==89)]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Mostra textualmente a tonalidade da música
mostraTonalidade:: [[Int]]->String
mostraTonalidade x=:[[acidentes,tonalidade]]
|lista == [0,0]="Do maior"
|lista == [0,1]="la menor"
|lista == [1,0]="Sol maior"
|lista == [1,1]="Mi menor"
|lista == [2,0]="RE maior"
|lista == [2,1]="Si menor"
|lista == [3,0]="La maior"
|lista == [3,1]="Fa# menor"
|lista == [4,0]="Mi maior"
|lista == [4,1]="Do# menor"
|lista == [5,0]="Si maior"
|lista == [5,1]="SoL# menor"
|lista == [6,0]="Fa# maior"
|lista == [6,1]="Re# menor"
|lista == [7,0]="Do# maior"
|lista == [7,1]="La# menor"

|lista == [255,0]="Fa maior"
|lista == [255,1]="Re menor"
|lista == [254,0]="Sib maior"
|lista == [254,1]="Sol menor"
|lista == [253,0]="MIb maior"
|lista == [253,1]="Do menor"
|lista == [252,0]="Lab maior"
```

```
|lista == [252,1]="Fa menor"
|lista == [251,0]="Reb maior"
|lista == [251,1]="Sib menor"
|lista == [250,0]="Solb maior"
|lista == [250,1]="Mib menor"
|lista == [249,0]="Dob maior"
|lista == [249,1]="Ab menor"
 |lista == [256,256]= "*Do maior*"
 |otherwise = ("Tonalidade desconhecida!"+++
            "\n MetaEvento  "+++ "[" +++ (toString acidentes)+++","
            +++(toString tonalidade)+++"]"+++"  nao catalogado")

where lista =: hd x

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/*  Função que calcula a tonalidade e o modo para bemóis. Quando for emol(>127),
o resultado do acidentevem multiplicado por 10. A dezena informa o número de
bemóis. Se for sustenido segue normalmente. */

calculaAcidentes :: [Int]-> [Int]
calculaAcidentes tonalidadeParaCalculo=:[acidente,modo]
|acidente > 127 = [(256 - acidente)*10,modo]//maior que 127 -> bemol
|otherwise = tonalidadeParaCalculo

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Função que pega a lista com o arquivo MIDI e o converte em uma lista com cada
track em uma lista */

SeparaTracks mTrks mThdMaisMTrks nTracks
     |nTracks == 0 = mThdMaisMTrks
     |otherwise =
      SeparaTracks (drop tamanhoTrack mTrks)( [take (tamanhoTrack) mTrks]++
          mThdMaisMTrks) (nTracks -1)
where
 tamanhoTrack =:(  ((mTrks!!5)*256*256)+ ((mTrks!!6)*256) + (mTrks!!7) +8 )

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// Rotina para separar um track em uma lista com listas de eventos

separaTrack trackLista = tl(reverse (SeparaEvento trackLista [[]]))
SeparaEvento trackLista listaFinal
     |(length trackLista) == 0 = listaFinal
     |otherwise = SeparaEvento
```

```
                (drop (hd evento) trackLista)
                [(tl evento)]++listaFinal
where
 evento =: lerEvento trackLista

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
/* A função  lerEvento a seguir, se encarrega de determinar o número de bytes do
deltatime, convertê-lo para ppq e identificar o evento, armazenando-o em uma lista
cuja cabeça é o número de bytes desta lista. Ela usa as funções listaMetaEvento...
Exemplo:
Start = lerEvento [129,2,255,47,4,1,2,3,4,126,143,60 ,100]
devolve [9,130,255,47,4,1,2,3,4], onde 9 é a quantidade de elementos desta nova
lista e 130 é o deltaTime convertido em ppq (129 e 2 = 129 -128 = 1. 1 = 128
(desloca um bit. para a direita)=>  128+2 = 130). Com 3 bytes fica:
Start = lerEvento [129,146,2,255,47,5,1,2,3,4,126,143,60 ,100]
devolve: [10,35074,255,47,5,1,2,3,4,126] */

lerEvento listaTracksMtrksSemCabecalho

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time
é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho == 255) =
            [(length(listaMetaEvento listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!3))):
            (listaMetaEvento listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!3))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time
é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento2 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!4))+1 ):

/* +1 devido ao contador ser a quantidade de bytes do evento inicial, como tem 2
bytes de delta time, temos uma contagem a mais para três bytes teremos duas
contagens a mais, ... */

            (listaMetaEvento2 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!4))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time
```

```
é de três bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==3) &&
        (next3 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento3 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!5))+2 ):
            (listaMetaEvento3 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!5))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time
é de quatro bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==4) &&
        (next4 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento4 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!6))+3 ):
            (listaMetaEvento4 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!6))]

/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144,
9F=159. E testa se o delta time é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 144)&&
        (next listaTracksMtrksSemCabecalho <= 159)=
            [length (take 4 listaTracksMtrksSemCabecalho):
            (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144,
9F=159. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 144)&&
        (next2 listaTracksMtrksSemCabecalho <= 159)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128, 8F
= 143. E testa se o delta time é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 128)&&
        (next listaTracksMtrksSemCabecalho <= 143)=
```

```
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128, 8F
= 143. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 128)&&
        (next2 listaTracksMtrksSemCabecalho <= 143)=
            [(length[(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
                (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 192)&&
        (next listaTracksMtrksSemCabecalho <= 207)=
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 192)&&
    (next2 listaTracksMtrksSemCabecalho <= 207)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]
/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume) B0 = 176, BF = 191. E
testa se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 176)&&
        (next listaTracksMtrksSemCabecalho <= 191)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
```

```
Exemplo: B7 = Volume Principal - MSB (Main Volume) B0 = 176, BF = 191. E
testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 176)&&
        (next2 listaTracksMtrksSemCabecalho <= 191)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

    | otherwise= []

//mensagem de pressão na tecla
/*A cláusula seguinte testa se o evento é um evento de mensagem de pressão na
tecla e testa se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 160)&&
        (next listaTracksMtrksSemCabecalho <= 175)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de pressão na tecla.
Exemplo: B7 = VolumeE testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 160)&&
        (next2 listaTracksMtrksSemCabecalho <= 175)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

    | otherwise= []

//fim mensagem pressão na tecla 2 bytes

//mensagem de variação do pitch bend 2 bytes

/*A cláusula seguinte testa se o evento é um evento variação do pitch bend e testa
se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 224)&&
```

```
        (next listaTracksMtrksSemCabecalho <= 239)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume) B0 = 176, BF = 191. E
testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 224)&&
        (next2 listaTracksMtrksSemCabecalho <= 239)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

// Para os demais eventos basta seguir os mesmos passos.
    | otherwise= []

//fim do pitch bend

//mensagem pressão no teclado 1 byte
/*A cláusula seguinte testa se o evento é um evento de pressão no teclado, e testa se
o delta time de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 208)&&
        (next listaTracksMtrksSemCabecalho <= 223)=
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]
/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time é de dois bytes */
    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 208)&&
    (next2 listaTracksMtrksSemCabecalho <= 223)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]

//fim de pressão no teclado

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Recebe uma lista, pega lista de metaevento, converte o eltaTime para ppq e
```

```
devolve a lista com o deltaTimePpq na cabeça e o metaEvento completo. */

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

listaMetaEvento listaTracksMtrksSemCabecalho tamanho  =
        take (tamanho + 4) listaTracksMtrksSemCabecalho

/* ListaMetaEvento devolve o metaEvento com cabeçalho e delta time (para delta
time = 1 byte) */

listaMetaEvento2 listaTracksMtrksSemCabecalho tamanho =
                 [(deltaTime listaTracksMtrksSemCabecalho):
                 (take (tamanho +3) (drop 2 listaTracksMtrksSemCabecalho))]

//listaMetaEvento2 = idem listaMetaEvento para deltaTime = 2 bytes
listaMetaEvento3  listaTracksMtrksSemCabecalho  tamanho  =
   [(deltaTime listaTracksMtrksSemCabecalho):
    (take (tamanho +3) (drop 3 listaTracksMtrksSemCabecalho))]

//listaMetaEvento3 = idem listaMetaEvento2 para deltaTime = 3 bytes

listaMetaEvento4  listaTracksMtrksSemCabecalho  tamanho  =
   [(deltaTime listaTracksMtrksSemCabecalho):
    (take (tamanho +3) (drop 4 listaTracksMtrksSemCabecalho))]
//listaMetaEvento4 = idem listaMetaEvento2 para deltaTime = 3 bytes

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

NBytesDeltaTime :: [Int] -> Int
NBytesDeltaTime lista  // antigo deltaB1
| (primeiroByte < 128)  = 1//UM BYTE
|( (primeiroByte > 127) &&
  (segundoByte  < 128) )= 2//DOIS BYTES
|( (primeiroByte > 127) &&
  (segundoByte  > 127) &&
  (terceiroByte < 128) )= 3 //TRÊS BYTES
|otherwise = 4 // quatro bytes
where
 primeiroByte =: lista!!0
 segundoByte  =: lista!!1
 terceiroByte =: lista!!2

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

//pega elementos de uma lista
```

```
next :: [Int]-> Int
next lista  = lista!!1

next2 :: [Int]-> Int
next2 lista = lista!!2

next3 :: [Int]-> Int
next3 lista = lista!!3

next4 :: [Int]-> Int
next4 lista = lista!!4

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

//esta função converte uma lista de bytes de deltaTime para ppq
deltaTime :: [Int]-> Int
deltaTime novaListaTracksMtrksSemCabecalho

//          Delta Time com 1 byte
|primeiroByte < 128 = primeiroByte // deltaTime4 com 1 byte

//          Delta Time com dois bytes
|(  primeiroByte > 127 &&
   segundoByte  < 128  ) =
   (primeiroByte bitand 64)/64 *2^13+
   (primeiroByte bitand 32)/32 *2^12+
   (primeiroByte bitand 16)/16 *2^11+
   (primeiroByte bitand 8)/8   *2^10+
   (primeiroByte bitand 4)/4   *2^9 +
   (primeiroByte bitand 2)/2   *2^8 +
   (primeiroByte bitand 1)/1   *2^7 +
   (segundoByte bitand 64)/64 *2^6 +
   (segundoByte bitand 32)/32 *2^5 +
   (segundoByte bitand 16)/16 *2^4 +
   (segundoByte bitand 8)/8 *2^3 +
   (segundoByte bitand 4)/4*2^2 +
   (segundoByte bitand 2)/2*2^1 +
   (segundoByte bitand 1)/1*2^0

//       Delta Time com três bytes
|(  primeiroByte > 127 &&
   segundoByte  > 127 &&
   terceiroByte < 128 ) =

   (primeiroByte bitand 64)/64 *2^20 +
```

```
   (primeiroByte bitand 32)/32 *2^19 +
   (primeiroByte bitand 16)/16 *2^18 +
   (primeiroByte bitand 8)/8 *2^17 +
   (primeiroByte bitand 4)/4*2^16 +
   (primeiroByte bitand 2)/2*2^15 +
   (primeiroByte bitand 1)/1*2^14 +

   (segundoByte bitand 64)/64 *2^13 +
   (segundoByte bitand 32)/32 *2^12 +
   (segundoByte bitand 16)/16 *2^11 +
   (segundoByte bitand 8)/8 *2^10 +
   (segundoByte bitand 4)/4*2^9  +
   (segundoByte bitand 2)/2*2^8  +
   (segundoByte bitand 1)/1*2^7  +
   (terceiroByte bitand 64)/64 *2^6 +
   (terceiroByte bitand 32)/32 *2^5 +
   (terceiroByte bitand 16)/16 *2^4 +
   (terceiroByte bitand 8)/8 *2^3 +
   (terceiroByte bitand 4)/4*2^2 +
   (terceiroByte bitand 2)/2*2^1 +
   (terceiroByte bitand 1)/1*2^0

//          Delta Time com quatro bytes
|(  primeiroByte > 127 &&
   segundoByte  > 127 &&
   terceiroByte > 127 &&
   quartoByte <128 ) =

   (primeiroByte bitand 64)/64 *2^27 +
   (primeiroByte bitand 32)/32 *2^26 +
   (primeiroByte bitand 16)/16 *2^25 +
   (primeiroByte bitand 8)/8 *2^24 +
   (primeiroByte bitand 4)/4*2^23 +
   (primeiroByte bitand 2)/2*2^22 +
   (primeiroByte bitand 1)/1*2^21 +

   (segundoByte bitand 64)/64 *2^20 +
   (segundoByte bitand 32)/32 *2^19 +
   (segundoByte bitand 16)/16 *2^18 +
   (segundoByte bitand 8)/8 *2^17 +
   (segundoByte bitand 4)/4*2^16+
   (segundoByte bitand 2)/2*2^15 +
   (segundoByte bitand 1)/1*2^14 +

   (terceiroByte bitand 64)/64 *2^13 +
```

```
    (terceiroByte bitand 32)/32 *2^12 +
    (terceiroByte bitand 16)/16 *2^11 +
    (terceiroByte bitand 8)/8 *2^10 +
    (terceiroByte bitand 4)/4*2^9+
    (terceiroByte bitand 2)/2*2^8 +
    (terceiroByte bitand 1)/1*2^7 +

    (quartoByte bitand 64)/64 *2^6 +
    (quartoByte bitand 32)/32 *2^5 +
    (quartoByte bitand 16)/16 *2^4 +
    (quartoByte bitand 8)/8 *2^3 +
    (quartoByte bitand 4)/4*2^2+
    (quartoByte bitand 2)/2*2^1 +
    (quartoByte bitand 1)/1*2^0

//          Mensagem de erro para deltatime > 4 bytes
|otherwise = abort "Nao existe deltaTime com mais de quarto bytes!"

//          Constantes utilizadas
where
      primeiroByte =: novaListaTracksMtrksSemCabecalho!!0
      segundoByte  =: novaListaTracksMtrksSemCabecalho!!1
      terceiroByte =: novaListaTracksMtrksSemCabecalho!!2
      quartoByte   =: novaListaTracksMtrksSemCabecalho!!3
```

Resultado obtido:

Do Maior

5.1.2.9. Fórmula de compasso:

A metodologia para se obter a fórmula de compasso segue a da tonalidade.

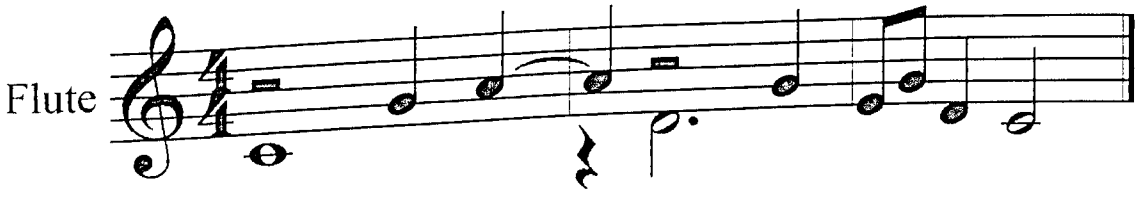


Figura 85

O programa:

```
module FormulaCompasso
import StdEnv, StdIO
```

```
Start comp

//para abrir o explorer
#    (maybeFile, comp)= selectInputFile comp
#    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
#arquivoLido = nomeArquivo
//fim da abertura do explorer

#    (ok, file, comp)= fopen nomeArquivo FReadText comp
#    (conteudo, file)= freads file 300000
#listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
#ppq =: ((listaTracks !! 12)*256) + (listaTracks !! 13)
#mThd =: take 14 listaTracks
#mTrks =: drop 14 listaTracks
#nTracks =: (mThd!!10)*256 + (mThd!!11)
#listaTracksMTrks =: (tl (reverse (SeparaTracks mTrks [mThd] nTracks)))
#listaTracksMtrksSemCabecalho =: map (drop 8) listaTracksMTrks
#listaEventosPorTrack =: map separaTrack listaTracksMtrksSemCabecalho
#TrackDeMetaEventos =: listaEventosPorTrack !! 0
#formulaDeCompasso =:VeSeTemFormulaDeCompasso  TrackDeMetaEventos
=formulaDeCompasso

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Rotina para verificar a existencia de fórmula de compasso se não tiver, ela
retorna 4/4. */

VeSeTemFormulaDeCompasso  trackDeMetaEventos
|fc == [] = [4,4]
|otherwise = fc
where
 fc  =:(  hd[[(x!!4),(2^(x!!5))]\\  x<-trackDeMetaEventos  |(x!!1  ==  255)  &&
(x!!2==88)]]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Função que pega a lista com o arquivo MIDI e o converte em uma lista com
cada track em uma lista */

SeparaTracks mTrks mThdMaisMTrks nTracks
     |nTracks == 0 = mThdMaisMTrks
     |otherwise =
      SeparaTracks (drop tamanhoTrack mTrks)( [take (tamanhoTrack) mTrks]++
             mThdMaisMTrks) (nTracks -1)
where
```

```
 tamanhoTrack =:(  ((mTrks!!5)*256*256)+ ((mTrks!!6)*256) + (mTrks!!7) +8 )

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Rotina para separar um track em uma lista com listas de eventos

separaTrack trackLista = tl(reverse (SeparaEvento trackLista [[]]))
SeparaEvento trackLista listaFinal
     |(length trackLista) == 0 = listaFinal
     |otherwise = SeparaEvento
                  (drop (hd evento) trackLista)
                  [(tl evento)]++listaFinal
where
  evento =: lerEvento trackLista

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* A função  lerevento a seguir, se encarrega de determinar o número de bytes do
deltatime, convertê-lo para ppq e identificar o evento, armazenando-o em uma
lista cuja cabeça é o número de bytes desta lista. Ela usa as funções
listaMetaEvento...
Exemplo:
Start = lerEvento [129,2,255,47,4,1,2,3,4,126,143,60 ,100]
devolve [9,130,255,47,4,1,2,3,4], onde 9 é a quantidade de elementos desta nova
lista e 130 é o deltaTime convertidoem ppq (129 e 2 = 129 -128 = 1. 1 =
128(desloca um bit.  para a direita)=>  128+2 = 130). Se tiver 3 bytes fica:
Start = lerEvento [129,146,2,255,47,5,1,2,3,4,126,143,60 ,100]
devolve: [10,35074,255,47,5,1,2,3,4,126]  */
lerEvento listaTracksMtrksSemCabecalho

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta
time é de um byte */

     |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
          (next listaTracksMtrksSemCabecalho == 255) =
               [(length(listaMetaEvento listaTracksMtrksSemCabecalho
               (listaTracksMtrksSemCabecalho!!3))):
               (listaMetaEvento listaTracksMtrksSemCabecalho
               (listaTracksMtrksSemCabecalho!!3))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta
time é de dois bytes */

     |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
          (next2 listaTracksMtrksSemCabecalho == 255)=
```

```
            [(length(listaMetaEvento2 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!4))+1 ):

/* +1 devido ao contador ser a quantidade de bytes do evento inicial, como tem 2
bytes de delta time, tem-se uma contagem a mais. Para três bytes tem-se duas
contagens a mais, ... */

            (listaMetaEvento2 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!4))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta
time é de três bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==3) &&
        (next3 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento3 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!5))+2 ):
            (listaMetaEvento3 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!5))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta
time é de quatro bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==4) &&
        (next4 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento4 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!6))+3 ):
            (listaMetaEvento4 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!6))]

/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144,
9F=159. E testa se o delta time é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 144)&&
        (next listaTracksMtrksSemCabecalho <= 159)=
            [length (take 4 listaTracksMtrksSemCabecalho):
            (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144,
9F=159. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 144)&&
        (next2 listaTracksMtrksSemCabecalho <= 159)=
```

```
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128,
8F = 143. E testa se o delta time é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 128)&&
        (next listaTracksMtrksSemCabecalho <= 143)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128,
8F = 143. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 128)&&
        (next2 listaTracksMtrksSemCabecalho <= 143)=
            [(length[(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
                (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 192)&&
        (next listaTracksMtrksSemCabecalho <= 207)=
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207, e testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 192)&&
    (next2 listaTracksMtrksSemCabecalho <= 207)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento controle (volume
```

```
geral,efeitos, sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 176)&&
        (next listaTracksMtrksSemCabecalho <= 191)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento controle (volume
geral,efeitos, sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191, e testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 176)&&
        (next2 listaTracksMtrksSemCabecalho <= 191)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

    | otherwise= []

//mensagem de pressão na tecla
/*A cláusula seguinte testa se o evento é um evento de mensagem de pressão na
tecla e testa se o delta time é de um byte. */
    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 160)&&
        (next listaTracksMtrksSemCabecalho <= 175)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de pressão na tecla Exemplo:
B7 = Volume, e testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 160)&&
        (next2 listaTracksMtrksSemCabecalho <= 175)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]
```

```
    | otherwise= []

//fim mensagem pressão na tecla 2 bytes

//mensagem de variação do pitch bend 2 bytes

/*A cláusula seguinte testa se o evento é um evento variação do pitch bend e testa
se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 224)&&
        (next listaTracksMtrksSemCabecalho <= 239)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento controle (volume
geral,efeitos, sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 224)&&
        (next2 listaTracksMtrksSemCabecalho <= 239)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

// Para os demais eventos basta seguir os mesmos passos.

    | otherwise= []

//fim do pitch bend

//mensagem pressão no teclado 1 byte

/*A cláusula seguinte testa se o evento é um evento de pressão no teclado. E testa
se o delta time de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 208)&&
        (next listaTracksMtrksSemCabecalho <= 223)=
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]
```

```
/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207, e testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 208)&&
    (next2 listaTracksMtrksSemCabecalho <= 223)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]

//fim de pressão no teclado

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Recebe uma lista, pega lista de metaevento, converte o deltaTime para ppq e
devolve a lista com o deltaTimePpq na cabeça e o metaEvento completo. */
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

listaMetaEvento listaTracksMtrksSemCabecalho tamanho =
        take (tamanho + 4) listaTracksMtrksSemCabecalho

/* listaMetaEvento devolve o metaEvento com cabeçalho e delta time (para delta
time = 1 byte) */

listaMetaEvento2 listaTracksMtrksSemCabecalho tamanho =
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take(tamanho+3) (drop 2 listaTracksMtrksSemCabecalho))]

//listaMetaEvento2 = idem listaMetaEvento para deltaTime = 2 bytes

listaMetaEvento3 listaTracksMtrksSemCabecalho tamanho =
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take (tamanho +3) (drop 3 listaTracksMtrksSemCabecalho))]

//listaMetaEvento3 = idem listaMetaEvento2 para deltaTime = 3 bytes

listaMetaEvento4 listaTracksMtrksSemCabecalho tamanho =
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take (tamanho +3) (drop 4 listaTracksMtrksSemCabecalho))]

//listaMetaEvento4 = idem listaMetaEvento2 para deltaTime = 3 bytes

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

NBytesDeltaTime :: [Int] -> Int
```

```
NBytesDeltaTime lista
| (primeiroByte < 128)  = 1 // um byte
|( (primeiroByte > 127) &&
  (segundoByte  < 128) )= 2 // dois bytes
|( (primeiroByte > 127) &&
  (segundoByte  > 127) &&
  (terceiroByte < 128) )= 3 // três bytes
|otherwise = 4 // quatro bytes

where
 primeiroByte =: lista!!0
 segundoByte  =: lista!!1
 terceiroByte =: lista!!2

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Pega elementos de uma lista

next :: [Int]-> Int
next lista  = lista!!1

next2 :: [Int]-> Int
next2 lista = lista!!2

next3 :: [Int]-> Int
next3 lista = lista!!3

next4 :: [Int]-> Int
next4 lista = lista!!4

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

//esta função converte uma lista de bytes de deltaTime para ppq

deltaTime :: [Int]-> Int
deltaTime novaListaTracksMtrksSemCabecalho

//          Delta Time com 1 byte
|primeiroByte < 128 = primeiroByte // deltaTime4 com 1 byte

//          Delta Time com dois bytes
|(  primeiroByte > 127 &&
   segundoByte  < 128  ) =
   (primeiroByte bitand 64)/64 *2^13+
   (primeiroByte bitand 32)/32 *2^12+
```

```
   (primeiroByte bitand 16)/16 *2^11+
   (primeiroByte bitand 8)/8   *2^10+
   (primeiroByte bitand 4)/4   *2^9 +
   (primeiroByte bitand 2)/2   *2^8 +
   (primeiroByte bitand 1)/1   *2^7 +

   (segundoByte bitand 64)/64 *2^6 +
   (segundoByte bitand 32)/32 *2^5 +
   (segundoByte bitand 16)/16 *2^4 +
   (segundoByte bitand 8)/8 *2^3 +
   (segundoByte bitand 4)/4*2^2 +
   (segundoByte bitand 2)/2*2^1 +
   (segundoByte bitand 1)/1*2^0

//       Delta Time com três bytes
|(  primeiroByte > 127 &&
   segundoByte  > 127 &&
   terceiroByte < 128 ) =

   (primeiroByte bitand 64)/64 *2^20 +
   (primeiroByte bitand 32)/32 *2^19 +
   (primeiroByte bitand 16)/16 *2^18 +
   (primeiroByte bitand 8)/8 *2^17 +
   (primeiroByte bitand 4)/4*2^16 +
   (primeiroByte bitand 2)/2*2^15 +
   (primeiroByte bitand 1)/1*2^14 +

   (segundoByte bitand 64)/64 *2^13 +
   (segundoByte bitand 32)/32 *2^12 +
   (segundoByte bitand 16)/16 *2^11 +
   (segundoByte bitand 8)/8 *2^10 +
   (segundoByte bitand 4)/4*2^9  +
   (segundoByte bitand 2)/2*2^8  +
   (segundoByte bitand 1)/1*2^7  +

   (terceiroByte bitand 64)/64 *2^6 +
   (terceiroByte bitand 32)/32 *2^5 +
   (terceiroByte bitand 16)/16 *2^4 +
   (terceiroByte bitand 8)/8 *2^3 +
   (terceiroByte bitand 4)/4*2^2 +
   (terceiroByte bitand 2)/2*2^1 +
   (terceiroByte bitand 1)/1*2^0

//           Delta Time com quatro bytes
|(  primeiroByte > 127 &&
```

```
    segundoByte  > 127 &&
    terceiroByte > 127 &&
    quartoByte <128 ) =

    (primeiroByte bitand 64)/64 *2^27 +
    (primeiroByte bitand 32)/32 *2^26 +
    (primeiroByte bitand 16)/16 *2^25 +
    (primeiroByte bitand 8)/8 *2^24 +
    (primeiroByte bitand 4)/4*2^23 +
    (primeiroByte bitand 2)/2*2^22 +
    (primeiroByte bitand 1)/1*2^21 +

    (segundoByte bitand 64)/64 *2^20 +
    (segundoByte bitand 32)/32 *2^19 +
    (segundoByte bitand 16)/16 *2^18 +
    (segundoByte bitand 8)/8 *2^17 +
    (segundoByte bitand 4)/4*2^16+
    (segundoByte bitand 2)/2*2^15 +
    (segundoByte bitand 1)/1*2^14 +
    (terceiroByte bitand 64)/64 *2^13 +
    (terceiroByte bitand 32)/32 *2^12 +
    (terceiroByte bitand 16)/16 *2^11 +
    (terceiroByte bitand 8)/8 *2^10 +
    (terceiroByte bitand 4)/4*2^9+
    (terceiroByte bitand 2)/2*2^8 +
    (terceiroByte bitand 1)/1*2^7 +

    (quartoByte bitand 64)/64 *2^6 +
    (quartoByte bitand 32)/32 *2^5 +
    (quartoByte bitand 16)/16 *2^4 +
    (quartoByte bitand 8)/8 *2^3 +
    (quartoByte bitand 4)/4*2^2+
    (quartoByte bitand 2)/2*2^1 +
    (quartoByte bitand 1)/1*2^0

//          Mensagem de erro para deltatime > 4 bytes
otherwise = abort "Nao existe deltaTime com mais de quarto bytes!"

//          Constantes utilizadas
where
      primeiroByte =: novaListaTracksMtrksSemCabecalho!!0
      segundoByte  =: novaListaTracksMtrksSemCabecalho!!1
      terceiroByte =: novaListaTracksMtrksSemCabecalho!!2
      quartoByte   =: novaListaTracksMtrksSemCabecalho!!3
```

Resultado obtido do exemplo inicial:

[4,4]

5.1.2.10. Instrumento do canal:

Esta ferramenta disponibiliza duas respostas diferentes:

- Em forma de uma lista contendo os tracks com o canal e o instrumento codificados em inteiro. Neste caso deve-se utilizar a função de saída **instrumentosLista**;
- Saída em forma textual por canal: Neste caso, deve-se utilizar a função de saída **NomeDeInstrumentos**;

No programa, a seguir, será mostrado o resultado obtido na forma textual

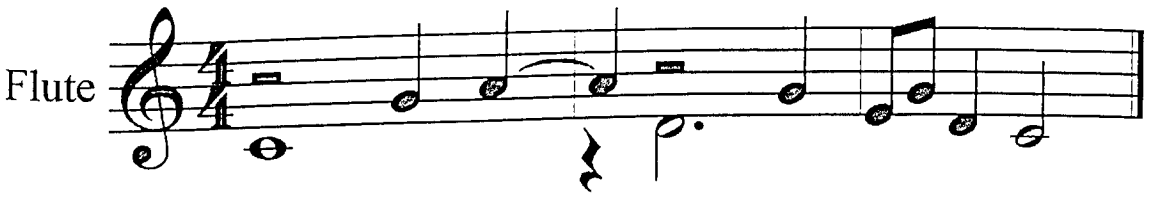


Figura 86

O programa:

```
module instrumentoCanal
import StdEnv, StdIO

Start comp

//para abrir o explorer
#    (maybeFile, comp)= selectInputFile comp
#    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
#arquivoLido = nomeArquivo
//fim da abertura do explorer

#    (ok, file, comp)= fopen nomeArquivo FReadText comp
#    (conteudo, file)= freads file 300000

#listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
```

```
#ppq =: ((listaTracks !! 12)*256) + (listaTracks !! 13)
#mThd =: take 14 listaTracks
#mTrks =: drop 14 listaTracks
#nTracks =: (mThd!!10)*256 + (mThd!!11)
#listaTracksMTrks =: (tl (reverse (SeparaTracks mTrks [mThd] nTracks)))
#listaTracksMtrksSemCabecalho =: map (drop 8) listaTracksMTrks
#listaEventosPorTrack =: map separaTrack listaTracksMtrksSemCabecalho
#TrackDeMetaEventos =: listaEventosPorTrack !! 0
#instrumentosLista =:[procuraIntrumentoEmUmTrack y \\ y<-listaEventosPorTrack ]
#NomeDeInstrumentos =:StringDeInstrumentos
        (map converteLinstToNomeInst(canalInst1 listaEventosPorTrack) )
=NomeDeInstrumentos

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Instrumento =: 192 // mudar instrumento canal zero
Salta1 =: (toString(toChar 13))+++(toString(toChar 10))

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
/* Outra forma mais direta de olhar os instrumentos por canal. Primeiro deve-se olhar
se o track possui metaEvento de instrumento. Se não tiver, devolve o número do
canal e instrumento piano. Se tiver, devolve o canal com o instrumento. */

canalInst1 listaEventosPorTrack =
    map canalInst (drop 1 listaEventosPorTrack)
canalInst  listaEventosPorTrack
|resultado == [] = resultado2 ++[0.0]
|otherwise = flatten[(map toReal (drop 1 x))\\ x<-listaEventosPorTrack|
    (((x !! 1)>= Instrumento )&&
    ((x !! 1)<= (Instrumento + 15)))]
where
    resultado2 =(map toReal [(hd(tl (listaEventosPorTrack!!0)))+48])
    resultado= flatten[drop 1 x\\ x<-listaEventosPorTrack|
    (((x !! 1)>= Instrumento )&&
    ((x !! 1)<= (Instrumento + 15)))]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

//NomeIntrumTracks x = map converteLinstToNomeInst x

converteLinstToNomeInst inst=:[x,y]
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 0.0) )   = "canal 10 = Percussão e Bateria - Standard Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 8.0) )   = "canal 10 = Percussão e Bateria - Room Set"
```

```
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 16.0) )  = "canal 10 = Percussão e Bateria - Power Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 48.0) )  = "canal 10 = Percussão e Bateria - Orchestral Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 32.0) )  = "canal 10 = Percussão e Bateria - Jazz Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 40.0) )  = "canal 10 = Percussão e Bateria - Brush Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 24.0) )  = "canal 10 = Percussão e Bateria - Electronic Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 56.0) )  = "canal 10 = Percussão e Bateria - SFX Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 25.0) )  = "canal 10 = Percussão e Bateria - TR-808 Set"
|((canalReal (x + 144.0 - 192.0)) == "canal 10")
            = "canal 10 = Percussão e Bateria - *Standard Set*"
|otherwise =(canalReal (x + 144.0 - 192.0))+++" = "+++
              (listaDeInstrumentosMIDI y)

StringDeInstrumentos x
|x == [] = ""
|otherwise =((hd x)+++ Salta1 +++ ( StringDeInstrumentos (tl x)))

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

canalReal :: Real -> {#Char}
canalReal canal = ["canal 1","canal 2","canal 3","canal 4","canal 5",
               "canal 6","canal 7","canal 8","canal 9","canal 10",
               "canal 11","canal 12","canal 13","canal 14",
               "canal 15","canal 16"]!!((toInt canal)-144)
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

listaDeInstrumentosMIDI :: Real -> {#Char}
listaDeInstrumentosMIDI indice =
    ["AcousticGrandPiano","BrightAcousticPiano",
    "ElectricGrandPiano","Honky-tonkPiano","ElectricPiano1",
    "ElectricPiano2","Harpsichord","Clavi","Celesta",
    "Glockenspiel","MusicBox","Vibraphone","Marimba",
    "Xylophone","TubularBells","Dulcimer","DrawbarOrgan",
    "PercussiveOrgan","RockOrgan","ChurchOrgan","ReedOrgan",
    "Accordion","Harmonica","TangoAccordion",
    "AcousticGuitar(nylon) ","AcousticGuitar(steel) ",
    "ElectricGuitar(jazz) ","ElectricGuitar(clean) ",
    "ElectricGuitar(muted) ","OverdrivenGuitar",
    "DistortionGuitar","Guitarharmonics","AcousticBass",
```

```
    "ElectricBass(finger)","ElectricBass(pick) ","FretlessBass",
    "SlapBass1","SlapBass2","SynthBass1","SynthBass2","Violin",
    "Viola","Cello","Contrabass","TremoloStrings",
    "PizzicatoStrings","OrchestralHarp","Timpani",
    "StringEnsemble1","StringEnsemble2","SynthStrings1",
    "SynthStrings2","ChoirAahs","VoiceOohs","SynthVoice",
    "OrchestraHit","Trumpet","Trombone","Tuba","MutedTrumpet",
    "FrenchHorn","BrassSection","SynthBrass1","SynthBrass",
    "SopranoSax","AltoSax","TenorSax","BaritoneSax","Oboe",
    "EnglishHorn","Bassoon","Clarinet","Piccolo","Flute",
    "Recorder","PanFlute","BlownBottle","Shakuhachi","Whistle",
    "Ocarina","Lead1(square)","Lead2(sawtooth)","Lead3(calliope)",
    "Lead4(chiff)","Lead5(charang)","Lead6(voice)","Lead7(fifths)",
    "Lead8(bass+lead)","Pad1(newage)","Pad2(warm)","Pad3(polysynth)",
    "Pad4(choir)","Pad5(bowed)","Pad6(metallic)","Pad7(halo)",
    "Pad8(sweep) ","FX1(rain) ","FX2(soundtrack) ","FX3(crystal) ",
    "FX4(atmosphere) ","FX5(brightness) ","FX6(goblins) ",
    "FX7(echoes) ","FX8(sci-fi) ","Sitar","Banjo","Shamisen","Koto",
    "Kalimba","Bagpipe","Fiddle","Shanai","TinkleBell","Agogo",
    "SteelDrums","Woodblock","TaikoDrum","MelodicTom","SynthDrum",
    "ReverseCymbal","GuitarFretNoise","BreathNoise","Seashore",
    "BirdTweet","TelephoneRing","Helicopter","Applause",
    "Gunshot"]!!(toInt indice)

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// lista de instrumentos

listaInstrumentos listaEventosPorTrack
   = [x\\ x<-listaEventosPorTrack |
     (((x !! 1)>= Instrumento )&&
     ((x !! 1)<= (Instrumento + 15)))]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Procura programChange em um track

procuraIntrumentoEmUmTrack lista =
  [tl x\\x<- lista
   | (x!!1 >= Instrumento) && (x!!1 <= Instrumento+15)]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Função que pega a lista com o arquivo MIDI e o converte em uma lista com cada
track em uma lista */
```

```
SeparaTracks mTrks mThdMaisMTrks nTracks
    |nTracks == 0 = mThdMaisMTrks
    |otherwise =
     SeparaTracks (drop tamanhoTrack mTrks)( [take (tamanhoTrack) mTrks]++
                  mThdMaisMTrks) (nTracks -1)
where
 tamanhoTrack =:(  ((mTrks!!5)*256*256)+ ((mTrks!!6)*256) + (mTrks!!7) +8 )
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Rotina para separar um track em uma lista com listas de eventos

separaTrack trackLista = tl(reverse (SeparaEvento trackLista [[]]))
SeparaEvento trackLista listaFinal
    |(length trackLista) == 0 = listaFinal
    |otherwise = SeparaEvento
                 (drop (hd evento) trackLista)
                 [(tl evento)]++listaFinal
where
 evento =: lerEvento trackLista

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* A função  lerEvento a seguir, se encarrega de determinar o número de bytes do
deltatime, convertê-lo para ppq e identificar o evento, armazenando-o em uma lista
cuja cabeça é o número de bytes desta lista. Ela usa as funções listaMetaEvento...
Exemplo:
Start = lerEvento [129,2,255,47,4,1,2,3,4,126,143,60 ,100]
devolve [9,130,255,47,4,1,2,3,4], onde 9 é a quantidade de elementos desta nova lista
e 130 é o deltaTime convertidoem ppq (129 e 2 = 129 -128 = 1. 1 = 128 (desloca um
bit. para a direita)=>  128+2 = 130). Se tiver 3 bytes, vamos ver como fica:
Start = lerEvento [129,146,2,255,47,5,1,2,3,4,126,143,60 ,100]
devolve: [10,35074,255,47,5,1,2,3,4,126] */

lerEvento listaTracksMtrksSemCabecalho

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é
de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho == 255) =
            [(length(listaMetaEvento listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!3))):
            (listaMetaEvento listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!3))]
```

```
/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é
de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!4))+1 ):

/* +1 devido ao contador ser a quantidade de bytes do evento inicial, como tem 2
bytes de delta time, temos uma contagem a mais para três bytes teremos duas
contagens a mais, ... */

            (listaMetaEvento2 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!4))]
/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é
de três bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==3) &&
        (next3 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento3 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!5))+2 ):
            (listaMetaEvento3 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!5))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é
de quatro bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==4) &&
        (next4 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento4 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!6))+3 ):
            (listaMetaEvento4 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!6))]

/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144, 9F=159.
E testa se o delta time é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 144)&&
        (next listaTracksMtrksSemCabecalho <= 159)=
            [length (take 4 listaTracksMtrksSemCabecalho):
            (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144, 9F=159.
E testa se o delta time é de dois bytes */
```

```
    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 144)&&
        (next2 listaTracksMtrksSemCabecalho <= 159)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128, 8F =
143. E testa se o delta time é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 128)&&
        (next listaTracksMtrksSemCabecalho <= 143)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128, 8F =
143. E testa se o delta time é de dois bytes  */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 128)&&
        (next2 listaTracksMtrksSemCabecalho <= 143)=
            [(length[(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
                (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 192)&&
        (next listaTracksMtrksSemCabecalho <= 207)=
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 192)&&
    (next2 listaTracksMtrksSemCabecalho <= 207)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
```

```
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 176)&&
        (next listaTracksMtrksSemCabecalho <= 191)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 176)&&
        (next2 listaTracksMtrksSemCabecalho <= 191)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

    | otherwise= []

//mensagem de pressão na tecla
/*A cláusula seguinte testa se o evento é um evento de mensagem de pressão na tecla
e testa se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 160)&&
        (next listaTracksMtrksSemCabecalho <= 175)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de pressão na tecla Exemplo: B7
= VolumeE testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 160)&&
        (next2 listaTracksMtrksSemCabecalho <= 175)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
```

```
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

    | otherwise= []

//fim mensagem pressão na tecla 2 bytes

//mensagem de variação do pitch bend 2 bytes

/*A cláusula seguinte testa se o evento é um evento variação do pitch bend e testa se
o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 224)&&
        (next listaTracksMtrksSemCabecalho <= 239)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191, e testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 224)&&
        (next2 listaTracksMtrksSemCabecalho <= 239)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

// Para os demais eventos basta seguir os mesmos passos

    | otherwise= []

//fim do pitch bend

//mensagem pressão no teclado 1 byte

/*A cláusula seguinte testa se o evento é um evento de pressão no teclado. E testa se
o delta time de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 208)&&
```

```
        (next listaTracksMtrksSemCabecalho <= 223)=
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207, e testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 208)&&
    (next2 listaTracksMtrksSemCabecalho <= 223)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]
//fim de pressão no teclado

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Recebe uma lista, pega lista de metaevento, converte o deltaTime para ppq e
devolve a lista com o deltaTimePpq na cabeça e o metaEvento completo. */

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

listaMetaEvento listaTracksMtrksSemCabecalho tamanho  = take
        (tamanho + 4) listaTracksMtrksSemCabecalho

/* listaMetaEvento devolve o metaEvento com cabeçalho e delta time (para delta
time = 1 byte) */

listaMetaEvento2 listaTracksMtrksSemCabecalho tamanho =
        [(deltaTime listaTracksMtrksSemCabecalho): (take (tamanho +3)
                (drop 2 listaTracksMtrksSemCabecalho))]

// listaMetaEvento2 = idem listaMetaEvento para deltaTime = 2 bytes

listaMetaEvento3  listaTracksMtrksSemCabecalho  tamanho  =
   [(deltaTime listaTracksMtrksSemCabecalho):
    (take (tamanho +3) (drop 3 listaTracksMtrksSemCabecalho))]

// listaMetaEvento3 = idem listaMetaEvento2 para deltaTime = 3 bytes

listaMetaEvento4  listaTracksMtrksSemCabecalho  tamanho  =
   [(deltaTime listaTracksMtrksSemCabecalho):
    (take (tamanho +3) (drop 4 listaTracksMtrksSemCabecalho))]

// listaMetaEvento4 = idem listaMetaEvento2 para deltaTime = 3 bytes
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

NBytesDeltaTime :: [Int] -> Int
NBytesDeltaTime lista
| (primeiroByte < 128)  = 1 // um byte
|( (primeiroByte > 127) &&

  (segundoByte  < 128) )= 2 // dois bytes
|( (primeiroByte > 127) &&
  (segundoByte  > 127) &&

  (terceiroByte < 128) )= 3 // três bytes
|otherwise = 4 // quatro bytes
where
 primeiroByte =: lista!!0
 segundoByte  =: lista!!1
 terceiroByte =: lista!!2

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Pega elementos de uma lista

next :: [Int]-> Int
next lista  = lista!!1

next2 :: [Int]-> Int
next2 lista = lista!!2
next3 :: [Int]-> Int
next3 lista = lista!!3

next4 :: [Int]-> Int
next4 lista = lista!!4

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//esta função converte uma lista de bytes de deltaTime para ppq

deltaTime :: [Int]-> Int
deltaTime novaListaTracksMtrksSemCabecalho

//          Delta Time com 1 byte
|primeiroByte < 128 = primeiroByte // deltaTime4 com 1 byte

//          Delta Time com dois bytes
|( primeiroByte > 127 &&
   segundoByte  < 128  ) =
```

```
  (primeiroByte bitand 64)/64 *2^13+
  (primeiroByte bitand 32)/32 *2^12+
  (primeiroByte bitand 16)/16 *2^11+
  (primeiroByte bitand 8)/8   *2^10+
  (primeiroByte bitand 4)/4   *2^9 +
  (primeiroByte bitand 2)/2   *2^8 +
  (primeiroByte bitand 1)/1   *2^7 +

  (segundoByte bitand 64)/64 *2^6 +
  (segundoByte bitand 32)/32 *2^5 +
  (segundoByte bitand 16)/16 *2^4 +
  (segundoByte bitand 8)/8 *2^3 +
  (segundoByte bitand 4)/4*2^2 +
  (segundoByte bitand 2)/2*2^1 +
  (segundoByte bitand 1)/1*2^0

//      Delta Time com três bytes
|( primeiroByte > 127 &&
  segundoByte  > 127 &&
  terceiroByte < 128 ) =

  (primeiroByte bitand 64)/64 *2^20 +
  (primeiroByte bitand 32)/32 *2^19 +
  (primeiroByte bitand 16)/16 *2^18 +
  (primeiroByte bitand 8)/8 *2^17 +
  (primeiroByte bitand 4)/4*2^16 +
  (primeiroByte bitand 2)/2*2^15 +
  (primeiroByte bitand 1)/1*2^14 +

  (segundoByte bitand 64)/64 *2^13 +
  (segundoByte bitand 32)/32 *2^12 +
  (segundoByte bitand 16)/16 *2^11 +
  (segundoByte bitand 8)/8 *2^10 +
  (segundoByte bitand 4)/4*2^9  +
  (segundoByte bitand 2)/2*2^8  +
  (segundoByte bitand 1)/1*2^7  +

  (terceiroByte bitand 64)/64 *2^6 +
  (terceiroByte bitand 32)/32 *2^5 +
  (terceiroByte bitand 16)/16 *2^4 +
  (terceiroByte bitand 8)/8 *2^3 +
  (terceiroByte bitand 4)/4*2^2 +
  (terceiroByte bitand 2)/2*2^1 +
  (terceiroByte bitand 1)/1*2^0
```

```
//          Delta Ttime com quatro bytes
|( primeiroByte > 127 &&
   segundoByte  > 127 &&
   terceiroByte > 127 &&
   quartoByte <128 ) =

   (primeiroByte bitand 64)/64 *2^27 +
   (primeiroByte bitand 32)/32 *2^26 +
   (primeiroByte bitand 16)/16 *2^25 +
   (primeiroByte bitand 8)/8 *2^24 +
   (primeiroByte bitand 4)/4*2^23 +
   (primeiroByte bitand 2)/2*2^22 +
   (primeiroByte bitand 1)/1*2^21 +

   (segundoByte bitand 64)/64 *2^20 +
   (segundoByte bitand 32)/32 *2^19 +
   (segundoByte bitand 16)/16 *2^18 +
   (segundoByte bitand 8)/8 *2^17 +
   (segundoByte bitand 4)/4*2^16+
   (segundoByte bitand 2)/2*2^15 +
   (segundoByte bitand 1)/1*2^14 +

   (terceiroByte bitand 64)/64 *2^13 +
   (terceiroByte bitand 32)/32 *2^12 +
   (terceiroByte bitand 16)/16 *2^11 +
   (terceiroByte bitand 8)/8 *2^10 +
   (terceiroByte bitand 4)/4*2^9+
   (terceiroByte bitand 2)/2*2^8 +
   (terceiroByte bitand 1)/1*2^7 +

   (quartoByte bitand 64)/64 *2^6 +
   (quartoByte bitand 32)/32 *2^5 +
   (quartoByte bitand 16)/16 *2^4 +
   (quartoByte bitand 8)/8 *2^3 +
   (quartoByte bitand 4)/4*2^2+
   (quartoByte bitand 2)/2*2^1 +
   (quartoByte bitand 1)/1*2^0

//          Mensagem de erro para deltatime > 4 bytes
|otherwise = abort "Nao existe deltaTime com mais de quarto bytes!"

//          Constantes utilizadas
where
    primeiroByte =: novaListaTracksMtrksSemCabecalho!!0
    segundoByte  =: novaListaTracksMtrksSemCabecalho!!1
```

```
terceiroByte =: novaListaTracksMtrksSemCabecalho!!2
quartoByte   =: novaListaTracksMtrksSemCabecalho!!3
```

Resultado obtido:

"canal 1 = Flute "

5.1.3. Leitura dos tracks musicais e conversão dos mesmos em formato texto

Esta é a última ferramenta e a que mais interessa aos futuros usuários dos programas (funções) geradas nos itens anteriores. Este item se encarrega de pegar a lista de eventos musicais e todas as informações musicais básicas identificados nos itens anteriores e gerar um arquivo textual equivalente. Este arquivo servirá para futuros parsers extraírem as informações desejadas para análise musical e atitudes quaisquer sobre os mesmos, tal, como exemplo, uma transposição musical e análise harmônica.

O maior trabalho desta ferramenta é realizar o *casting* (conversão de tipo) entre as listas de inteiros e reais para uma string (formato texto).

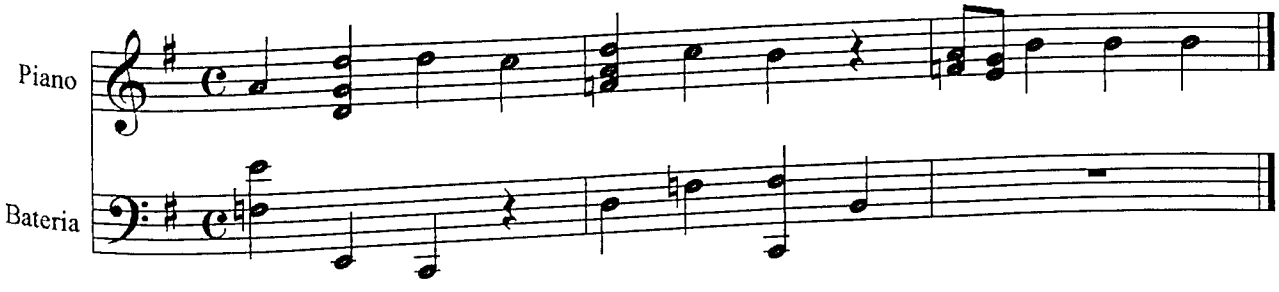


Figura 87

O programa

```
module MIDITools
import StdEnv,MIDIBiblioteca,StdIO

Start comp
//para abrir o explorer

#    (maybeFile, comp)= selectInputFile comp
|    isNothing maybeFile= comp // Se for Nothing, terminou.
#    (Just nomeArquivo)= maybeFile //Retira nome de arquivo
```

```
#arquivoLido = nomeArquivo
//fim da abertura do explorer

#    (ok, file, comp)= fopen nomeArquivo FReadText comp
#    (conteudo, file)= freads file 300000

#listaTracks =: [(digitToInt x)+48\\ x<-:conteudo]
#formato =: listaTracks !! 9
#ppq =: ((listaTracks !! 12)*256) + (listaTracks !! 13)
#mThd =: take 14 listaTracks
#mTrks =: drop 14 listaTracks
#nTracks =: (mThd!!10)*256 + (mThd!!11)
#listaTracksMTrks =: (tl (reverse (SeparaTracks mTrks [mThd] nTracks)))
#listaTracksMtrksSemCabecalho =: map (drop 8) listaTracksMTrks
#listaEventosPorTrack =: map separaTrack listaTracksMtrksSemCabecalho
#listaEventosPorTrackSemTrack1 =: drop 1 listaEventosPorTrack
#SoListadeltaTimes =:map ListadeltaTimes listaEventosPorTrack
#SoListaEventos =:map ListaEventos listaEventosPorTrack
#separaDeltaDeEvento =:flatten (separaDeltaTimeLista listaEventosPorTrack)
#ListaDeltaTimesPorTrack =: [map hd x\\x<-SoListadeltaTimes]
#ListaDeTempoDeCadaTrack =: map sum ListaDeltaTimesPorTrack
#TrackDeMetaEventos =: listaEventosPorTrack !! 0
//#metaFormulaDeCompasso =:VeSeTemFormulaDeCompasso
        hd[[(x!!4),(2^(x!!5))]\\ x<-TrackDeMetaEventos |
        (x!!1 == 255) && (x!!2==88)]

#metaFormulaDeCompasso =:VeSeTemFormulaDeCompasso TrackDeMetaEventos

#metronomo =: [60000000/(((x!!4)*256*256)+((x!!5)*256)+(x!!6))  \\
        x<-TrackDeMetaEventos | (x!!1 == 255) && (x!!2 ==81)]
#tonalidadeEscrita=:  mostraTonalidade (veSeTem TrackDeMetaEventos)

/*  Devolve o codigo da tonalidade. Se for bemol, tem casa da dezena no
primeiro elemento da tupla: Ex. normal -> (1,0) um sustenido e tom maior
Ex. bemol -> (10,0) um bemol e  tom maior. */

#tonalidadeParaCalculo =: calculaAcidentes(hd (veSeTem TrackDeMetaEventos))
#conversaoMinutos =: (toReal (hd metronomo))/ (3600.0 * (toReal ppq))
#conversaoSegundos =: (toReal (hd metronomo))/ (60.0 * (toReal ppq))
#tempoMusicaPorTrackEmMinutos =: map ( (*)conversaoMinutos)
                        (map toReal ListaDeTempoDeCadaTrack)
#tempoMusicaPorTrackEmSegundos =: map ( (*)conversaoSegundos)
                        (map toReal ListaDeTempoDeCadaTrack)
#instrumentosLista =:[procuraIntrumentoEmUmTrack y \\ y<-listaEventosPorTrack ]
#metaEventosListacompleto =:
```

```
 [procuraMetaEventoEmUmTrack y \\ y<-listaEventosPorTrack ]
#eliminaInstrumentosLista =:
 [EliminaIntrumentoEmUmTrack y \\ y<-listaEventosPorTrackSemTrack1 ]
#eliminaMetaEventoscompleto =:
 [EliminaMetaEventoEmUmTrack y \\ y<-listaEventosPorTrackSemTrack1 ]
#eliminaControlecompleto =:
 [EliminacontroleEmUmTrack y \\ y<-listaEventosPorTrackSemTrack1 ]
#eliminaTudoMenosAtivaEdesativaNotas =:
   [EliminaTudoMenosNota y \\ y <- listaEventosPorTrackSemTrack1]
#separaDeltaDeSomenteEventos =:
    flatten (separaDeltaTimeLista eliminaTudoMenosAtivaEdesativaNotas)
#listaDeAtivarEdesativarNotas =: eliminaTudoMenosAtivaEdesativaNotas
#listaDeAtivarEdesativarNotasReal =: listIntToReal listaDeAtivarEdesativarNotas

#TracksNotasTempo =: map (notaTempo (toReal ppq))
            listaDeAtivarEdesativarNotasReal

#listaTipoMusicaAtivarDesativarNotas =:  aplicaNOTAS2 TracksNotasTempo

#TracksAcordesTempoComSEparador1000 =:
    map separaAcordesEmUmTrack listaTipoMusicaAtivarDesativarNotas

#TracksApenasAcoresTempo = Elimina1000DaMusica
        TracksAcordesTempoComSEparador1000
#listaTipoMusicaAtivarDesativarNotasTexto=:
        mudaParaTextoMusicaNtracks
        listaTipoMusicaAtivarDesativarNotas
#TracksAcordesTempoComSEparador1000Texto =:
        map separaAcordesEmUmTrackTexto
        listaTipoMusicaAtivarDesativarNotasTexto

#TracksApenasAcoresTempoTexto =
        Elimina1000DaMusicaTexto
        TracksAcordesTempoComSEparador1000Texto
#MusicaTextoSemCompasso =MusicaTexto TracksApenasAcoresTempoTexto

#AcordesEnotas =:separaMusicaTexto MusicaTextoSemCompasso
#formulaDeCompasso =:VeSeTemFormulaDeCompasso  TrackDeMetaEventos
#InstrumentosPorCanal =:InstrumentosNome listaEventosPorTrack
#NomeDeInstrumentos =:StringDeInstrumentos
        (map converteLinstToNomeInst(canalInst1 listaEventosPorTrack) )
#resultado =: [[(Dados [("ppq= "+++(toString ppq)),
              ("Key= " +++(toString (formulaDeCompasso!!0))
                    +++ "/"
                    +++(toString (formulaDeCompasso!!1)) ),
```

```
            ("tonalidade= " +++ tonalidadeEscrita),
            ("canais= "+++(toString (nTracks -1))),
            ("metronomo= "+++(toString (hd metronomo)))])]]++
            AcordesEnotas

#fim =:

"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    +++Salta1 +++
    "    Tradutor de Arquivos MIDI para Texto Utilizando"
    +++Salta1 +++
    "              Linguagem Funcional CLEAN"
    +++Salta1+++

    "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    +++Salta1+++
    "    Convertendo MIDI FORMATO 1 para TEXTO"
    +++Salta1+++
    "    Faculdade de Engenharia Elétrica"
    +++Salta1+++
    "    Universidade Federal de Uberlândia"
    +++Salta2+++

    "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
        +++ Salta1+++
        "Arquivo MIDI: " +++ arquivoLido
                    +++ Salta1+++
        "Resultado da conversão em: " +++ arquivoLido +++ ".txt"+++Salta1+++

    "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
                    +++Salta1+++"Autor: André Campos Machado"+++Salta1+++
                    "Orientador: Luciano Vieira Lima"+++Salta1+++

    "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
                +++ Salta1 +++
        "              INFORMAÇÕES GERAIS"+++Salta1+++

    "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
        +++ Salta2+++

        "Número de canais= " +++ (toString (nTracks-1))
                        +++ Salta1 +++
     "ppq = " +++(toString ppq)
                            +++ Salta1+++
     "Fórmula de Compasso= "+++(toString (formulaDeCompasso!!0))
```

```
                    +++ "/"
                    +++(toString (formulaDeCompasso!!1))+++ " ,"
                    +++Salta1 +++
    "Tonalidade da música= " +++ tonalidadeEscrita
                     +++ Salta1 +++
    "Metrônomo= " +++(toString (hd metronomo))
                +++ Salta2 +++
    "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
                +++ Salta1 +++
      "              INSTRUMENTOS DE CADA CANAL " +++ Salta1+++
   "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
                   +++ Salta2+++
        NomeDeInstrumentos +++ Salta1 +++
    "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
                +++ Salta1 +++
                (NotaAcordeMultiTrack(drop 1 resultado)) +++ Salta2

#   (ok, resultadoFinal, comp)= fopen (arquivoLido +++ ".txt")FWriteText comp
#   resultadoFinal= fwrites fim resultadoFinal
#   (ok, comp)= fclose resultadoFinal comp

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// Saída gráfica

#   (ok, comp)= fclose file comp
#   (ids, comp)= openIds 2 comp

= startIO NDI 1 (dialogo ids fim) [ProcessClose closeProcess] comp

Salta1 =: (toString(toChar 13))+++(toString(toChar 10))
Salta2 =: Salta1 +++ Salta1
Salta1Linha =: {toChar 13,toChar 10}
Salta2Linhas =: Salta1Linha +++ Salta1Linha

//fim da saída gráfica

//          TESTES DO START

//= TracksAcordesTempoComSEparador1000
//= TracksApenasAcoresTempo
//= mudaParaTextoMusicaNtracks listaTipoMusicaAtivarDesativarNotas
//= listaTipoMusicaAtivarDesativarNotas
//=TracksAcordesTempoComSEparador1000Texto
//=MusicaTextoSemCompasso
//= MusicaTexto TracksApenasAcoresTempoTexto
```

```
//=calculaTempo "1.5"
//=listaTipoMusicaAtivarDesativarNotasTexto
// 5 = listaTipoMusicaAtivarDesativarNotas
// 4 =listaTipoMusicaAtivarDesativarNotasTexto
// 3 =TracksAcordesTempoComSEparador1000Texto
// 2 =MusicaTextoSemCompasso
//=separaMusicaTexto MusicaTextoSemCompasso
//=AcordesEnotas
//=resultado
//=InstrumentosPorCanal
//=comp
//=listaEventosPorTrack
//=map converteLinstToNomeInst(canalInst1 listaEventosPorTrack)
//=length InstrumentosPorCanal
//=InstrumentosReais listaEventosPorTrack
//= map converteLinstToNomeInst (InstrumentosReais listaEventosPorTrack)
//= InstrumentosNome listaEventosPorTrack
//=StringDeInstrumentos InstrumentosPorCanal
//=metronomo
//=formulaDeCompasso
//= (tempoStringParaReal "sm" )+(tempoStringParaReal "m")+(tempoStringParaReal
"mp")
/*=PegaTempoDoAcordeTexto (Acorde [NOTA
        ["canal 1","do5","Volume = 80","1sm"],
        NOTA ["canal 1","la5","Volume = 80","1"] ])
*/
//=PegaTempoDaNotaTexto c
//= metaFormulaDeCompasso
//=numeroDeNotasDoCompasso TrackDeMetaEventos

//XXXXXXXXXXXXXXXXX FIM DO TESTE DO START XXXXXXXXXXXXX

//              Constantes

AtivarNota=:144 //ativar nota canal zero
AtivarNotaReal=:144.0
DesativarNota =:128 //desativar nota canal zero
DesativarNotaReal =:128.0
Instrumento =:192 // mudar instrumento canal zero
Comando =:176 //controle no canal zero
MetaEvento =: 255

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Funções da interface
```

```
dialogo [windId, edId] texto pst
#    (error, pst)= openDialog 1 eD pst
= pst
where
    eD= Dialog "Conversor MIDI -> Texto" cmp
            [    WindowClose xP,
                 WindowId windId]
    cmp= EditControl (texto%(0, 70000)) (PixelWidth 440) 30
            [    ControlId edId]
    xP (ls, ps)= (ls, closeProcess ps)

// Fim das funções gráficas

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Rotinas para transformar inteiro em strings dando um sentido musical ao conteúdo
do arquivo. */

//              FUNÇÕES DE BIBLIOTECA
/*
tempoReal :: Real -> {#Char}
volumeReal :: Real -> {#Char}
canalReal :: Real -> {#Char}
notaCifraRealBemol :: Real -> {#Char}
notaCifraRealSus :: Real -> {#Char}
*/

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Outra forma mais direta de olhar os instrumentos por canal. Primeiro deve-se olhar
se o track possui metaEvento de instrumento se não tiver, devolve o número do canal e
instrumento piano se tiver, devolve o canal com o instrumento. */

canalInst1 listaEventosPorTrack =
    map canalInst (drop 1 listaEventosPorTrack)
canalInst  listaEventosPorTrack
|resultado == [] = resultado2 ++[0.0]
|otherwise = flatten[(map toReal (drop 1 x))\\ x<-listaEventosPorTrack|
    (((x !! 1)>= Instrumento )&&
    ((x !! 1)<= (Instrumento + 15)))]
where
     resultado2 =(map toReal [(hd(tl (listaEventosPorTrack!!0)))+48])
     resultado= flatten[drop 1 x\\ x<-listaEventosPorTrack|
    (((x !! 1)>= Instrumento )&&
    ((x !! 1)<= (Instrumento + 15)))]
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

converteLinstToNomeInst inst=:[x,y]
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 0.0) )  = "canal 10 = Percussão e Bateria - Standard Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 8.0) )  = "canal 10 = Percussão e Bateria - Room Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 16.0) )  = "canal 10 = Percussão e Bateria - Power Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 48.0) )  = "canal 10 = Percussão e Bateria - Orchestral Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 32.0) )  = "canal 10 = Percussão e Bateria - Jazz Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 40.0) )  = "canal 10 = Percussão e Bateria - Brush Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 24.0) )  = "canal 10 = Percussão e Bateria - Electronic Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 56.0) )  = "canal 10 = Percussão e Bateria - SFX Set"
|(((canalReal (x + 144.0 - 192.0)) == "canal 10")&&
 (y == 25.0) )  = "canal 10 = Percussão e Bateria - TR-808 Set"
|((canalReal (x + 144.0 - 192.0)) == "canal 10")
            = "canal 10 = Percussão e Bateria - *Standard Set*"
|otherwise =(canalReal (x + 144.0 - 192.0))+++" = "+++ (listaDeInstrumentosMIDI y)

StringDeInstrumentos x
|x == [] = ""
|otherwise =((hd x)+++ Salta1 +++ ( StringDeInstrumentos (tl x)))

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Lista de instrumentos

listaInstrumentos listaEventosPorTrack
   = [x\\ x<-listaEventosPorTrack |
     (((x !! 1)>= Instrumento )&&
     ((x !! 1)<= (Instrumento + 15)))]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Checa estrutura: Quem é quem?

:: Musica3 = Acorde [Musica2] | Nota [Musica2]|Dados [{#Char}]
:: Musica2 = NOTA [{#Char}]
:: Musica =  NOTAS [Real]
```

```
checaNotaOuAcorde y=:(Nota x) = "Nota"
checaNotaOuAcorde y=:(Acorde x)=  "Acorde"
checaNotaOuAcorde y=:(Dados x)=  "Dados"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Conversão canais texto

verificaCanalNotaOuAcorde y=:(Acorde [(x=:(NOTA z)):w])= hd z
verificaCanalNotaOuAcorde y=:(Nota [NOTA z]) = hd z
testaNotaAcorde x
|checaNotaOuAcorde x == "Nota"   = converteNota   x
|checaNotaOuAcorde x == "Acorde" = converteAcorte x
|otherwise = "errei"

converteNota x=:(Nota [y])=converteNotaTextoPre y
converteNotaTextoPre y=:(NOTA w)
|length w > 3 = converteNotaTexto y
|otherwise = convertePausa y
converteNotaTexto y=:(NOTA [canal,nota,volume,figura])=
  "       Nota=[("+++nota+++","+++figura+++"),"
         +++volume+++"]"+++Salta1
convertePausa y=:(NOTA [x,z]) =
   "        Nota=[(" +++x +++","+++ z +++")]"+++ Salta1

converteAcorte x=:(Acorde y) = converteAcordeTexto  y
    ("       Acorde="+++Salta1) 1
converteAcordeTexto y stringFinal cont
|length y == 0 = stringFinal
|size stringFinal < 2 = converteAcordeTexto
      (drop 1 y)
      (stringFinal +++ "                   "
              +++(converteNotaTextoPre2 (hd y)))
      (cont+1)
|otherwise = converteAcordeTexto
      (drop 1 y)
      (stringFinal +++ "                   "
              +++(converteNotaTextoPre2 (hd y)))
      (cont+1)
converteNotaTextoPre2 y=:(NOTA w)
|pausaOuNota w == "Nota" = converteNotaTexto2 y
|pausaOuNota w == "Pausa" = convertePausa2 y
pausaOuNota x
|hd x == "Pausa" = "Pausa"
|otherwise = "Nota"
```

```
converteNotaTexto2 y=:(NOTA [canal,nota,volume,figura])=
  "[("+++nota+++","+++figura+++"),"
        +++volume+++"]"+++Salta1
convertePausa2 y=:(NOTA [x,z]) =
     "[(" +++x +++","+++ z +++")]"+++ Salta1

concatStringList x = unificaStringLista x
   (verificaCanalNotaOuAcorde (hd x) /*"CANAL"*/ +++Salta2)
unificaStringLista x stringFinal
|length x == 0 = stringFinal+++Salta1
|otherwise = unificaStringLista
           (drop 1 x)
           (stringFinal+++(testaNotaAcorde(hd x)))

NotaAcordeMultiTrack x = unificaMulti x

     ("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
     +++Salta1 +++
      "              GRADE DA ORQUESTRA"+++Salta1+++
      "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"+++
                                              Salta2)


unificaMulti x trackFinal
|length x == 0 = trackFinal
|otherwise = unificaMulti
                  (drop 1 x)
                  (trackFinal+++concatStringList (hd x))

// Fim da conversão texto canais

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

coloca x y = [x:y]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// Rotina para verificar a existencia de fórmula de compasso se não tiver, retorna 4/4.

VeSeTemFormulaDeCompasso  trackDeMetaEventos
|fc == [] = [4,4]
|otherwise = fc
where
 fc   =:(   hd[[(x!!4),(2^(x!!5))]\\   x<-trackDeMetaEventos   |(x!!1   ==   255)   &&
(x!!2==88)])

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
// Rotina para ver quantas notas entram no compasso

numeroDeNotasDoCompasso x = (toReal(fcomp!!0)) * (4.0/(toReal(fcomp!!1)))
where
 fcomp =: VeSeTemFormulaDeCompasso x

/* Para separar o compasso é o seguinte: pegar todas as notas enquanto a soma das
notas for menor que o número de notas por compasso. */

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Batera x
|( (x < 35)||(x > 82)) = "none"
| otherwise =["Acoustic Bass Drum","Bass Drum 1","Side Stick",
 "Acoustic Snare","Hand Clap","Electric Snare",
 "Low Floor Tom","Closed Hi Hat","High Floor Tom",
 "Pedal Hi-Hat","Low Tom","Open Hi-Hat","Low-Mid Tom",
 "Hi Mid Tom","Crash Cymbal 1","High Tom","Ride Cymbal 1",
 "Chinese Cymbal","Ride Bell","Splash Cymbal","Cowbell",
 "Crash Cymbal 2","Vibraslap","Ride Cymbal 2","Hi Bongo",
 "Low Bongo","Mute Hi Conga","Open Hi Conga","Low Conga",
 "High Timbale","Low Timbale","High Agogo","Low Agogo",
 "Cabasa","Maracas","Short Whistle","Long Whistle",
 "Short Guiro","Long Guiro","Claves","Hi Wood Block",
 "Low Wood Block","Mute Cuica","Open Cuica","Mute Triangle",
 "Open Triangle"]!!(x-35)

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

TrocaRealParaString y=:(NOTAS [dt,canal,nota,volume,tempo])
|canal==1000.0 =
  (NOTA [toString dt,toString canal,notaCifraRealSus nota,
       volumeReal volume,calculaTempo tempo])
|canal ==2000.0 =
  (NOTA ["Pausa", calculaTempo tempo])
|canal == 153.0 =(NOTA [toString dt,canalReal canal,
      (notaCifraRealSus nota)+++" - "+++ (Batera (toInt nota)),
       volumeReal volume,calculaTempo tempo])
|otherwise =
  (NOTA [toString dt,canalReal canal,notaCifraRealSus nota,
       volumeReal volume,calculaTempo tempo])

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

//                    SEPARAR COMPASSOS
```

```
/ Ppega o tempo em uma nota
// ESTRUTURA-> (Nota [(NOTA ["canal 1","re5","Volume = 80","1sm"])])

PegaTempoDaNotaTexto y=:(Nota [NOTA z]) = hd(reverse z)
PegaCanalDaNotaTexto y=:(Nota [NOTA z]) = hd z

/*pega o tempo em um acorde (supõe-se uma só voz, ou seja,  que as notas do acorde
possuam o mesmo tempo) */
/* ESTRUTURA->
(Acorde
  [(NOTA ["canal 1","do5","Volume = 80","1sm"]),
   (NOTA ["canal 1","la5","Volume = 80","1sm"])
  ]) */

PegaTempoDoAcordeTexto y=:(Acorde [(x=:(NOTA z)):w])= hd(reverse z)
PegaCanalDoAcordeTexto y=:(Acorde [(x=:(NOTA z)):w])= hd z

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Elimina 1000 da lista

Elimina1000EmUmTrack y = [z\\ z<-y |(length z) > 1 || ((el1000 z)!!1)<> 1000.0]
Elimina1000EmUmTrackTexto y = [z\\ z<-y |(length z) > 1 ||
                  ((el1000Texto z)!!1)<> "1000"]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//          Separa Acordes de Notas

separaMusicaTexto y = map separaDeUmTrackTexto y
separaDeUmTrackTexto y = map separaDeUmaListaTexto y
separaDeUmaListaTexto y
|length y >1 = Acorde y
|otherwise = Nota y

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

MusicaTexto mt = map tiradeUmTrackMusicaTexto mt
tiradeUmTrackMusicaTexto w = map tiraDaListappqTexto w
tiraDaListappqTexto y  = [tiraDeUmTextoppq2 z\\z<-y]
tiraDeUmTextoppq y=:(NOTA [dt,canal,nota,volume,tempo]) =
  (NOTA [canal,nota,volume,tempo])

tiraDeUmTextoppq2 y=:(NOTA z)
 |(length z > 2) =(NOTA (drop 1 z))
 |otherwise = y
```

```
Elimina1000DaMusica y = [(Elimina1000EmUmTrack z)\\z<-y]
Elimina1000DaMusicaTexto y = [(Elimina1000EmUmTrackTexto z)\\z<-y]
el1000 k=:[NOTAS x] = x
el1000Texto k=: [NOTA x] = x
mudaParaTextoMusicaNtracks y = map mudaParaTextoMusicaTrack y
mudaParaTextoMusicaTrack y = [TrocaRealParaString z \\ z<- y]

//    Converte os inteiros para tipo MUSICA

aplicaNOTAS1 :: [[Real]] -> [Musica]
aplicaNOTAS1 l1 = [(NOTAS y)\\ y<-l1]
aplicaNOTAS2:: [[[Real]]] -> [[Musica]]
aplicaNOTAS2 l2 = map aplicaNOTAS1 l2

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

//                  MODIFICANDO ROTINAS TIPO MUSICA

montaAcordes :: [Musica]-> [Musica]
montaAcordes lista
|testaAtinvarNotaRealTipoMusica (lista!!0)==True =
          ((takeWhile testaAtinvarNotaRealTipoMusica lista))
|otherwise = [lista!!0]

montaAcordesTexto :: [Musica2]-> [Musica2]
montaAcordesTexto lista
|testaAtinvarNotaRealTipoMusicaTexto (lista!!0)==True =
    ((takeWhile testaAtinvarNotaRealTipoMusicaTexto lista))
|otherwise = [lista!!0]

igual1000 lista=:[NOTAS x]
|x!!1 == 1000.0 = False
|otherwise = True

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
tiraDasOutras lista=:[NOTAS x] = x

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Testa ativar nota do tipo MUSICA

testaAtinvarNotaRealTipoMusica ::  Musica -> Bool
testaAtinvarNotaRealTipoMusica nota =:(NOTAS lista)
| (((lista !! 1)>= AtivarNotaReal )&&
    ((lista !! 1)<= (AtivarNotaReal + 15.0)))= True
```

```
|otherwise = False

testaAtinvarNotaRealTipoMusicaTexto ::  Musica2 -> Bool
testaAtinvarNotaRealTipoMusicaTexto y=:(NOTA z)
|(((z!!1)== "1000")||((z!!0) == "Pausa")) = False
|otherwise = True

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

separaAcordesEmUmTrack :: [Musica] -> [[Musica]]
separaAcordesEmUmTrack lista = separaAcordesEmUmTrackAux lista
                                        [[NOTAS [0.0]]]

separaAcordesEmUmTrackAux :: [Musica] [[Musica]] -> [[Musica]]
separaAcordesEmUmTrackAux lista listaFinal
| length lista == 0 = drop 1 (reverse listaFinal)
| otherwise =
   separaAcordesEmUmTrackAux
      (drop (length (montaAcordes lista)) lista)
       ([(montaAcordes lista)]++ listaFinal)

separaAcordesEmUmTrackTexto :: [Musica2] -> [[Musica2]]
separaAcordesEmUmTrackTexto lista =
          separaAcordesEmUmTrackAuxTexto lista [[NOTA [""]]]

separaAcordesEmUmTrackAuxTexto :: [Musica2] [[Musica2]] -> [[Musica2]]
separaAcordesEmUmTrackAuxTexto lista listaFinal
| length lista == 0 = drop 1 (reverse listaFinal)
| otherwise =
   separaAcordesEmUmTrackAuxTexto
      (drop (length (montaAcordesTexto lista)) lista)
       ([(montaAcordesTexto lista)]++ listaFinal)

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Testa se os eventos de ativar e desativar nota são do mesmo canal e se é a mesma
nota */

IgualAtivarDesativar lista1 lista2
|((lista1!!1) == ((lista2!!1) + 16.0)) && ((lista1!!2) == (lista2!!2)) = True
|otherwise = False

diferenteAtivarDesativar lista1 lista2
|((lista1!!1)<> ((lista2!!1) + 16.0)) = True
|((lista1!!2) <> (lista2!!2)) = True
```

```
|otherwise = False

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Soma deltaTimes das listas e incorpora à primeira

atualizaTempo l1=:[x1:y1] l2=:[x2:y2]= [(x1+x2):y1]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
converte1 l1IntToReal = [toReal y\\ y<-l1IntToReal]
converte2 l2IntToReal = map converte1 l2IntToReal
listIntToReal x = [converte2 y\\y<-x]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
/* Pega o evento de ativar nota e procura o desativar, o tempo da nota é a soma de
todos os deltaTime existentes até encontrar o desativar */

notaTempo :: Real [[Real]]->[[Real]]
notaTempo ppq lista = igualOUnao ppq lista []

igualOUnao :: Real [[Real]] [[Real]] -> [[Real]]
igualOUnao ppq lista listaFinal
|lista == [] = reverse listaFinal
|otherwise =
   igualOUnao
      ppq
        (testa1000drop lista)
        (testa1000 lista ppq listaFinal)
where
 metade1 =: (takeWhile (diferenteAtivarDesativar (FirstLista lista)) lista)
 metade2 =: (dropWhile (diferenteAtivarDesativar (FirstLista lista)) lista)

testa1000drop lista
|(FirstLista  lista)!!1 == 1000.0 = drop 1 lista
|otherwise = (drop 1 (metade1 ++  [insertAt 1 1000.0 ((take 1 metade2)!!0)] ++
         (drop 1 metade2)))
where
 metade1 =: (takeWhile (diferenteAtivarDesativar (FirstLista lista)) lista)
 metade2 =: (dropWhile (diferenteAtivarDesativar (FirstLista lista)) lista)

testa1000 lista ppq listaFinal
|((FirstLista  lista)!!1 == 1000.0)||((FirstLista  lista)!!1 == 2000.0) =
         ([(FirstLista  lista)]++ listaFinal)
|(FirstLista  lista)!!0 == 0.0 =
                  (poeNaLista
```

```
            (novoTempoFim
              (FirstLista  lista)
              (tempoAtualizado metade1 metade2 ppq)
             )
             listaFinal)

|otherwise =
 [(novoTempoFim  (FirstLista  lista)
          (tempoAtualizado2 metade1 metade2 ppq))]++
          [mudaFirstLista (FirstLista lista) ppq]++
             listaFinal
where
 metade1 =: (takeWhile (diferenteAtivarDesativar (FirstLista lista)) lista)
 metade2 =: (dropWhile (diferenteAtivarDesativar (FirstLista lista)) lista)
mudaFirstLista x=:[tempo,comando,nota,volume] ppq =
            [tempo,2000.0,nota,volume,tempo/ppq]
novoTempoFim :: [Real] Real -> [Real]
novoTempoFim lista z  = lista ++ [z]

// Pega um elemento do desativar nota e acrescenta 62
//Exemplo: 80H = 128 passa a ser 1000; 8FH = 133 passa a ser 205

mudaDesativarPara19X  lista =
  (primeiroLL lista)++(segundoLL lista )++(restoLL lista)

primeiroLL lista = [(take 1 lista)!!0]
segundoLL lista =  [((take 1 lista)!!0) + 62.0]
restoLL lista = drop 2 (take 1 lista)

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

//funções auxiliares para atualizar tempo de nota. Calcula o tempo da nota
tempoAtualizado metade1 metade2 ppq  = (  (  sum[(x!!0)\\x<- metade1 ]  )+
        ((metade2!!0)!!0)  )/ppq
tempoAtualizado2 metade1 metade2 ppq = (  (  sum[(x!!0)\\x<- metade1 |
        (seraAtivarNota x)]  )+((metade2!!0)!!0)  )/ppq
where
 seraAtivarNota lista
  | /*(((lista!!1)<>2000.0)&&*/ ((lista!!1)< 144.0) /*)*/ =True
  |otherwise = False

//coloca o tempo da nota na lista

novoTempo (lista=:[x:y]) z = [z:y]
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

poeNaLista x lista = [x:lista]
FirstLista lista = lista!!0
NextLista lista  = lista!!1

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Testa se alista é um determinado evento, se for devolve a lista com o código do
evento e qual é. Se não for, devolve a lista vazia. */

testaEvento evento lista
| (((lista !! 1)>= evento )&&
    ((lista !! 1)<= (evento + 15)))= [lista!!1,lista!!2]
|otherwise = []
//testa um evento, se for, devolve True, caso contrário, devolve False

testaAtinvarNotaReal :: [Real] -> Bool
testaAtinvarNotaReal lista
| (((lista !! 1)>= AtivarNotaReal )&&
    ((lista !! 1)<= (AtivarNotaReal + 15.0)))= True
|otherwise = False

testaAtinvarNota:: [Int] -> Bool
testaAtinvarNota lista
| (((lista !! 1)>= AtivarNota )&&
    ((lista !! 1)<= (AtivarNota + 15)))= True
|otherwise = False

testaDestivarNotaReal :: [Real] -> Bool
testaDestivarNotaReal lista
| (((lista !! 1)>= DesativarNotaReal )&&
    ((lista !! 1)<= (DesativarNotaReal + 15.0)))= True
|otherwise = False
testaDestivarNota :: [Int] -> Bool
testaDestivarNota lista
| (((lista !! 1)>= DesativarNota )&&
    ((lista !! 1)<= (DesativarNota + 15)))= True
|otherwise = False

testaMetaEvento :: [Int] -> Bool
testaMetaEvento lista
|(lista !! 1)== 255 = True
|otherwise = False
```

```
testaProgramChange :: [Int] -> Bool
testaProgramChange lista
| (((lista !! 1)>= Instrumento )&&
    ((lista !! 1)<= (Instrumento + 15)))= True
|otherwise = False

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Para testar eliminar ....
VerSeEliminaInstrumento lista =[procuraIntrumentoEmUmTrack y \\ y<-lista ]
VerSeEliminaMetaEvento lista =[procuraMetaEventoEmUmTrack y \\ y<-lista ]
VerSeEliminaComando lista =[procuraComandoEmUmTrack y \\ y<-lista ]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

      // Procura programChange em um track
procuraIntrumentoEmUmTrack lista =
 [tl x\\x<- lista
  | (x!!1 >= Instrumento) && (x!!1 <= Instrumento+15)]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

         // Procura metaEventos em um track
procuraMetaEventoEmUmTrack lista =
 [x\\x<- lista
  | (x!!1 == MetaEvento)]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

      // Procura programChange em um track
procuraComandoEmUmTrack lista =
 [tl x\\x<- lista
  | (x!!1 >= Comando) && (x!!1 <= Comando+15)]
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

EliminaTudoMenosNota lista=
 EliminacontroleEmUmTrack
  (EliminaMetaEventoEmUmTrack
   (EliminaIntrumentoEmUmTrack lista)
  )

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

                  // Elimina programChange em um track
EliminaIntrumentoEmUmTrack lista =
```

```
 [x\\x<- lista
  | (x!!1 < Instrumento) || (x!!1 > Instrumento+15)]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

          // Elimina metaEventos em um track
EliminaMetaEventoEmUmTrack lista =
 [x\\x<- lista
  | (x!!1 <> MetaEvento)]
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

                 // Elimina programChange em um track
EliminacontroleEmUmTrack lista =
 [x\\x<- lista
  | (x!!1 < Comando) || (x!!1 > Comando+15)]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// Localiza metaEvento tonalidade (nem todos os software têm)
veSeTem trackDeMetaEventos
//devolve [[256,256]] se não tiver meta evento tonalidade,
//como é o caso do Encore, se tiver, devolve a lista [[acid, modo]]
|[x\\x<-trackDeMetaEventos |
    (x!!1 == 255) &&
    (x!!2==89)]== [] = [[256,256]]
//TONALIDADE
|[x\\x<-trackDeMetaEventos |
    (x!!1 == 255) &&
    (x!!2==89)]<> [] =
        [[(x!!4), (x!!5)]\\x<-trackDeMetaEventos |
            (x!!1 == 255) &&
            (x!!2==89)]
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ListadeltaTimes listaEventosPorTrack = [[hd x]\\ x<-listaEventosPorTrack]
ListaEventos listaEventosPorTrack = [tl x\\ x<-listaEventosPorTrack]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

separaDeltaTimeLista listaEventosPorTrack =
  [map serparaCabecaCauda x \\ x<-listaEventosPorTrack]

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

serparaCabecaCauda [lista:cauda] =[[lista],cauda]
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Função que pega a lista com o arquivo MIDI e o converte em uma lista com cada
track em uma lista. */

SeparaTracks mTrks mThdMaisMTrks nTracks
    |nTracks == 0 = mThdMaisMTrks
    |otherwise =
     SeparaTracks (drop tamanhoTrack mTrks)( [take (tamanhoTrack) mTrks]++
    mThdMaisMTrks) (nTracks -1)
where
 tamanhoTrack =:(  ((mTrks!!5)*256*256)+ ((mTrks!!6)*256) + (mTrks!!7) +8 )

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

/* Recebe uma lista, pega lista de metaevento, converte o deltaTime para ppq e
devolve a lista com o deltaTimePpq na cabeça e o metaEvento completo. */

listaMetaEvento listaTracksMtrksSemCabecalho tamanho  = take (tamanho + 4)
        listaTracksMtrksSemCabecalho

/* ListaMetaEvento devolve o metaEvento com cabeçalho e delta time (para delta time
= 1 byte). */

listaMetaEvento2 listaTracksMtrksSemCabecalho tamanho = [(deltaTime
        listaTracksMtrksSemCabecalho):
        (take (tamanho +3) (drop 2 listaTracksMtrksSemCabecalho))]

//listaMetaEvento2 = idem listaMetaEvento para deltaTime = 2 bytes

listaMetaEvento3  listaTracksMtrksSemCabecalho  tamanho  =
  [(deltaTime listaTracksMtrksSemCabecalho):
   (take (tamanho +3) (drop 3 listaTracksMtrksSemCabecalho))]

//listaMetaEvento3 = idem listaMetaEvento2 para deltaTime = 3 bytes

listaMetaEvento4  listaTracksMtrksSemCabecalho  tamanho  =
  [(deltaTime listaTracksMtrksSemCabecalho):
   (take (tamanho +3) (drop 4 listaTracksMtrksSemCabecalho))]

//listaMetaEvento4 = idem listaMetaEvento2 para deltaTime = 3 bytes

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
/* A função  lerEvento a seguir, se encarrega de determinar o número de bytes do
deltatime, convertê-lo para ppq e identificar o evento, armazenando-o em uma lista
```

```
cuja cabeça é o número de bytes desta lista. Ela usa as funções listaMetaEvento...
Exemplo:
Start = lerEvento [129,2,255,47,4,1,2,3,4,126,143,60 ,100]
devolve [9,130,255,47,4,1,2,3,4], onde 9 é a quantidade de elementos
desta nova lista e 130 é o deltaTime convertidoem ppq (129 e 2 = 129 -128 = 1. 1 =
128(desloca um bit. para a direita)=>  128+2 = 130). Se tiver 3 bytes, fica:
Start = lerEvento [129,146,2,255,47,5,1,2,3,4,126,143,60 ,100]
devolve: [10,35074,255,47,5,1,2,3,4,126]  */
lerEvento listaTracksMtrksSemCabecalho

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é de
um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho == 255) =
            [(length(listaMetaEvento listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!3))):
            (listaMetaEvento listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!3))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é de
dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento2 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!4))+1 ):

/* +1 devido ao contador ser a quantidade de bytes do evento inicial, como tem 2 bytes
de delta time, tem-se uma contagem a mais para três bytes tem-se duas contagens a
mais, ... */
            (listaMetaEvento2 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!4))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é de
três bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==3) &&
        (next3 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento3 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!5))+2 ):
            (listaTracksMtrksSemCabecalho
            (listaMetaEvento3 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!5))]

/*A cláusula seguinte testa se o evento é um metaEvento FF=255 e se o delta time é de
```

```
quatro bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==4) &&
        (next4 listaTracksMtrksSemCabecalho == 255)=
            [(length(listaMetaEvento4 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!6))+3 ):
            (listaMetaEvento4 listaTracksMtrksSemCabecalho
            (listaTracksMtrksSemCabecalho!!6))]
/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144, 9F=159. E
testa se o delta time é de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 144)&&
        (next listaTracksMtrksSemCabecalho <= 159)=
            [length (take 4 listaTracksMtrksSemCabecalho):(take 4
            listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de ativar nota 90 = 144, 9F=159. E
testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 144)&&
        (next2 listaTracksMtrksSemCabecalho <= 159)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128, 8F =
143. E testa se o delta time é de um byte */
    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 128)&&
        (next listaTracksMtrksSemCabecalho <= 143)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de desativar nota 80 = 128, 8F =
143. E testa se o delta time é de dois bytes  */
    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 128)&&
        (next2 listaTracksMtrksSemCabecalho <= 143)=
            [(length[(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
                (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]
```

```
/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 192)&&
        (next listaTracksMtrksSemCabecalho <= 207)=
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 192)&&
    (next2 listaTracksMtrksSemCabecalho <= 207)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 176)&&
        (next listaTracksMtrksSemCabecalho <= 191)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]
/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 176)&&
        (next2 listaTracksMtrksSemCabecalho <= 191)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho)]+1):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho):
            [(deltaTime listaTracksMtrksSemCabecalho)]]
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

// Para os demais eventos basta seguir os mesmos passos

    | otherwise= []
```

```
// Mensagem de pressão na tecla
/*A cláusula seguinte testa se o evento é um evento de mensagem de pressão na tecla e
testa se o delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 160)&&
        (next listaTracksMtrksSemCabecalho <= 175)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de pressão na tecla Exemplo: B7 =
Volume. E testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 160)&&
        (next2 listaTracksMtrksSemCabecalho <= 175)=
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

// Para os demais eventos basta seguir os mesmos passos
    | otherwise= []

// Fim mensagem pressão na tecla 2 bytes

// Mensagem de variação do pitch bend 2 bytes

/*A cláusula seguinte testa se o evento é um evento variação do pitch bend e testa se o
delta time é de um byte. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 224)&&
        (next listaTracksMtrksSemCabecalho <= 239)=
            [length (take 4 listaTracksMtrksSemCabecalho):
                (take 4 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento controle (volume geral,efeitos,
sustain...)
Exemplo: B7 = Volume Principal - MSB (Main Volume)
B0 = 176, BF = 191. E testa se o delta time é de dois bytes. */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
        (next2 listaTracksMtrksSemCabecalho >= 224)&&
        (next2 listaTracksMtrksSemCabecalho <= 239)=
```

```
            [(length [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]+1):
            [(deltaTime listaTracksMtrksSemCabecalho):
            (take 3 (drop 2 listaTracksMtrksSemCabecalho))]]

// Para os demais eventos basta seguir os mesmos passos
    | otherwise= []

// Fim do pitch bend

// Mensagem pressão no teclado 1 byte

/*A cláusula seguinte testa se o evento é um evento de pressão no teclado. E testa se o
delta time de um byte */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==1) &&
        (next listaTracksMtrksSemCabecalho >= 208)&&
        (next listaTracksMtrksSemCabecalho <= 223)=
            [(length(take 3 listaTracksMtrksSemCabecalho)):
                (take 3 listaTracksMtrksSemCabecalho)]

/*A cláusula seguinte testa se o evento é um evento de program change (mudar o
instrumento) C0 = 192, CF = 207. E testa se o delta time é de dois bytes */

    |((NBytesDeltaTime listaTracksMtrksSemCabecalho)==2) &&
    (next2 listaTracksMtrksSemCabecalho >= 208)&&
    (next2 listaTracksMtrksSemCabecalho <= 223)=
        [(length(take 3 listaTracksMtrksSemCabecalho)+1):
        [(deltaTime listaTracksMtrksSemCabecalho):
        (take 2 (drop 2 listaTracksMtrksSemCabecalho))]]

// Fim de pressão no teclado

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
// Rotina para separar um track em uma lista com listas de eventos
separaTrack trackLista = tl(reverse (SeparaEvento trackLista [[]]))
SeparaEvento trackLista listaFinal
    |(length trackLista) == 0 = listaFinal
    |otherwise = SeparaEvento
                (drop (hd evento) trackLista)
                [(tl evento)]++listaFinal
where
 evento =: lerEvento trackLista
```

O Resultado obtido é representado pelo quadro abaixo

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    Ferramentas didáticas para manipulação de arquivos MIDI
            em linguagem funcional CLEAN
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    Convertendo MIDI FORMATO 1 para TEXTO
     Faculdade de Engenharia Elétrica
     Universidade Federal de Uberlândia
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Arquivo MIDI: D:\André\Tese\Defesa\batera1.mid
Resultado da conversão em: D:\ André\Tese\Defesa\batera1.MID.txt
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Autor: André Campos Machado
Orientador: Luciano Vieira Lima
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            INFORMAÇÕES GERAIS
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Número de canais= 2
ppq = 240
Fórmula de Compasso= 4/4 ,
Tonalidade da música= *Do maior*
Metrônomo= 100
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            INSTRUMENTOS DE CADA CANAL
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
canal 1 = AcousticGrandPiano
canal 10 = Percussão e Bateria - Standard Set
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            GRADE DA ORQUESTRA
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 canal 1

     Nota=[(la5,sm),Volume = 80]
     Acorde=
          [(re6,sm),Volume = 80]
          [(sol5,sm),Volume = 80]
          [(re5,sm),Volume = 80]
     Nota=[(re6,sm),Volume = 80]
     Nota=[(do6,sm),Volume = 80]
     Acorde=
          [(la5,sm),Volume = 80]
          [(re6,sm),Volume = 80]
```

```
        [(fa5,sm),Volume = 80]
    Nota=[(do6,sm),Volume = 80]
    Nota=[(si5,sm),Volume = 80]
    Nota=[(Pausa,sm)]
    Acorde=
        [(la5,c),Volume = 80]
        [(fa5,sm),Volume = 80]
    Acorde=
        [(sol5,c),Volume = 80]
        [(mi5,sm),Volume = 80]
    Nota=[(si5,sm),Volume = 80]
    Nota=[(si5,sm),Volume = 80]
    Nota=[(si5,sm),Volume = 80]

canal 10

    Acorde=
        [(mi5 - High Timbale,sm),Volume = 80]
        [(fa4 - Ride Bell,sm),Volume = 80]
    Nota=[(mi3 - Electric Snare,sm),Volume = 80]
    Nota=[(do3 - Bass Drum 1,sm),Volume = 80]
    Nota=[(Pausa,sm)]
    Nota=[(re4 - High Tom,sm),Volume = 80]
    Nota=[(fa4 - Ride Bell,sm),Volume = 80]
    Acorde=
        [(fa4 - Ride Bell,sm),Volume = 80]
        [(do3 - Bass Drum 1,sm),Volume = 80]
    Nota=[(si3 - Low-Mid Tom,sm),Volume = 80]
```

Capítulo 6

6.1. Conclusões

Este estudo teve como foco principal sanar a carência de material didático que permitisse um auto-aprendizado das técnicas e princípios básicos de geração de ferramentas para manipulação de arquivos MIDI SMF formato 1, utilizando linguagens funcionais e suas potencialidades. O mesmo conclui por demonstrar o quão aderente é o código gerado nas implementações das ferramentas em linguagem funcional CLEAN. Estas ferramentas associadas aos conceitos e detalhes ao protocolo MIDI aqui apresentados com detalhes, permitirão aos usuários e pesquisadores em computação sônica implementar, com relativa facilidade, seus próprios sistemas de análise musical, seqüenciamento, editoração de partituras e outras atividades no domínio musical.

Os programas gerados e aqui denominados de ferramentas de manipulação de SMF formato 1, comprovam que a escolha de uma linguagem funcional, no caso CLEAN, foi uma decisão acertada ao se mostrar fortemente aderente ao paradigma modelado. O fato da música e seus protocolos serem estritamente matemáticos, e, também, dos dados manipulados serem facilmente representados, modelados por listas de eventos, concluem por tornar os programas gerados extremamente velozes, já que estes tipos de dados são eficientemente manipulados por esta linguagem.

Os objetivos descritos inicialmente, portanto, foram plenamente satisfeitos, abrindo uma perspectiva de novos trabalhos, tanto para especialistas em computação, como para os músicos que trabalhem com computação sônica.

6.2. Trabalhos Futuros

Conforme afirmado anteriormente e comprovado pelos capítulos apresentados neste estudo, partindo-se das ferramentas implementadas, torna-se, com pouco esforço, elaborar uma nova gama de projetos no domínio musical.

Como metas a serem desenvolvidas em curto prazo podemos citar os seguintes projetos:

- De imediato, o sistema, ao ler os arquivos MIDI e convertê-lo em formato texto, pode servir para o treinamento do sistema de composição automática baseada em estilo [10];
- A partir do conhecimento disponibilizado do protocolo MIDI, já está em andamento um novo estudo na Universidade Federal de Uberlândia, também em linguagem funcional, que manipule qualquer formato de arquivos SMF, com ou sem Running Status, o qual culminará pela elaboração de um aplicativo geral de manipulação de SMFs;
- Análise musical a partir de um arquivo MIDI SMF;
- Programas de transposição musical do arquivo MIDI SMF;
- Criar interfaces gráficas para geração e manipulação de partituras;
- Implementação de um sistema automático de composição por contraponto.

Anexos

A.1. MIDIBiblioteca.icl

```
implementation module MIDIBiblioteca
import StdEnv

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
//                Testes da biblioteca
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

//Start= ppqToDeltaTime 237191324
//Start= deltaTime [241,141,129,28]
//Start = mostraTonalidade [[255,1]]
//Start = calculaAcidentes [255,1]
//Start = NBytesDeltaTime [129,128,64,32,45]
//Start = Primeiro [129,128,64,32,45]
//Start = next[129,128,64,32,45]
//Start = next2 [129,128,64,32,45]
//Start = next3 [129,128,64,32,45]
//Start =notaCifraIntSus 60

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
/*função que transforma ppq em uma lista de bytes de deltaTimes*/
ppqToDeltaTime :: Int -> [Int]
```

```
ppqToDeltaTime x
|x<=127 = [x]
|otherwise = ppqToDT (x/128)  [(x rem 128)]
ppqToDT x valorDT
 |x <= 127 =  [(x+128):valorDT]
 |x >= 128 =  ppqToDT ( x/128 ) [((x rem 128)+128):valorDT]

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
//CONVERTE NOTA_INTEIRO PARA NOTA_CIFRA

notaCifraRealSus :: Real -> {#Char}
notaCifraRealSus indiceReal =
  ["do0","do#0","re0","re#0","mi0","fa0","fa#0","sol0","sol#0","la0","la#0","si0",
  "do1","do#1","re1","re#1","mi1","fa1","fa#1","sol1","sol#1","la1","la#1","si1",
  "do2","do#2","re2","re#2","mi2","fa2","fa#2","sol2","sol#2","la2","la#2","si2",
  "do3","do#3","re3","re#3","mi3","fa3","fa#3","sol3","sol#3","la3","la#3","si3",
  "do4","do#4","re4","re#4","mi4","fa4","fa#4","sol4","sol#4","la4","la#4","si4",
  "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","do#7","re7","re#7","mi0","fa7","fa#7","sol7","sol#7","la7","la#7","si7",
  "do8","do#8","re8","re#8","mi8","fa8","fa#8","sol8","sol#8","la8","la#8","si8",
  "do9","do#9","re9","re#9","mi9","fa9","fa#9","sol9","sol#9","la9","la#9","si9",
 "do10","do#10","re10","re#10","mi10","fa10","fa#10","sol10","sol#10","la10","la#10
    ","si10"]!!(toInt indiceReal)

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

volumeReal :: Real -> {#Char}
volumeReal indiceReal = ("Volume = " +++ (toString indiceReal))
```

```
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
tempoReal :: Real -> {#Char}
tempoReal indiceReal = ((toString indiceReal)+++"sm")
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

calculaTempo :: Real ->{#Char}
calculaTempo tempo
|(tempo < (4.0/3.0*1.1))&& (tempo > (4.0/3.0*0.9) ) = "q32m"
|(tempo < (0.6666667*1.1))&& (tempo > (0.6666667*0.9) ) = "q32sm"
|(tempo < (0.3333333*1.1))&& (tempo > (0.3333333*0.9) ) = "q32c"
|(tempo < (0.166666667*1.1))&& (tempo > (0.166666667*0.9) ) = "q32sc"
|(tempo < (0.75*1.1))&& (tempo > (0.75*0.9) ) ="cp"
|(tempo < (0.25*1.1))&& (tempo > (0.25*0.9) ) ="sc"
|(tempo < (0.875*1.1))&& (tempo > (0.875*0.9) ) ="cpp"
|(tempo < (0.125*1.1))&& (tempo > (0.125*0.9) ) ="f"
|(tempo < (0.375*1.1))&& (tempo > (0.375*0.9) ) ="scp"
|(tempo < (0.4375*1.1))&& (tempo > (0.4375*0.9) ) ="scpp"
|(tempo < (0.0625*1.1))&& (tempo > (0.0625*0.9) ) ="sf"
|(tempo < (1.0*1.1))&& (tempo > (1.0*0.9) ) ="sm"
|(tempo < (1.5*1.1))&& (tempo > (1.5*0.9) ) ="smp"
|(tempo < (1.75*1.1))&& (tempo > (1.75*0.9) ) ="smpp"
|(tempo < (0.5*1.1))&& (tempo > (0.5*0.9) ) ="c"
|(tempo < (2.0*1.1))&& (tempo > (2.0*0.9) ) ="m"
|(tempo < (2.5*1.1))&& (tempo > (2.5*0.9) ) ="mc"
|(tempo < (3.0*1.1))&& (tempo > (3.0*0.9) ) ="mp"
|(tempo < (3.5*1.1))&& (tempo > (3.5*0.9) ) ="mpp"
|(tempo < (4.0*1.1))&& (tempo > (4.0*0.9) ) ="sb"
|(tempo < (5.0*1.1))&& (tempo > (5.0*0.9) ) ="sbsm"
|(tempo < (6.0*1.1))&& (tempo > (6.0*0.9) ) ="sbp"
|(tempo < (7.0*1.1))&& (tempo > (7.0*0.9) ) ="sbpp"
```

```
|(tempo < (8.0*1.1))&& (tempo > (8.0*0.9) ) ="b"
|otherwise = toString tempo

tempoStringParaReal :: {#Char} -> Real
tempoStringParaReal tempo
|tempo == "b" = 8.0
|tempo == "sb" = 4.0
|tempo == "sbp" = 6.0
|tempo == "sbpp" = 7.0
|tempo == "m" = 2.0
|tempo == "mp" = 3.0
|tempo == "mpp" = 3.5
|tempo == "sm" = 1.0
|tempo == "smp" = 1.5
|tempo == "smpp" = 1.75
|tempo == "c" = 0.5
|tempo == "cp" = 0.75
|tempo == "cpp" = 0.875
|tempo == "sc" = 0.25
|tempo == "scp" = 0.375
|tempo == "scpp" = 0.4375
|tempo == "f" = 0.125
|tempo == "sf" = 0.0625
|tempo == "q32m" = 4.0/3.0
|tempo == "q32sm" =2.0/3.0
|tempo == "q32c" = 1.0/3.0
|tempo == "q32sc" = 0.5/3.0
|otherwise = toReal tempo

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
notaCifraRealBemol :: Real -> {#Char}
notaCifraRealBemol indiceReal =
 ["do0","reb0","re0","mib0","mi0","fa0","solb0","sol0","lab0","la0","sib0","si0",
  "do1","reb1","re1","mib1","mi1","fa1","solb1","sol1","lab1","la1","sib1","si1",
  "do2","reb2","re2","mib2","mi2","fa2","solb2","sol2","lab2","la2","sib2","si2",
  "do3","reb3","re3","mib3","mi3","fa3","solb3","sol3","lab3","la3","sib3","si3",
  "do4","reb4","re4","mib4","mi4","fa4","solb4","sol4","lab4","la4","sib4","si4",
  "do5","reb5","re5","mib5","mi5","fa5","solb5","sol5","lab5","la5","sib5","si5",
  "do6","reb6","re6","mib6","mi6","fa6","solb6","sol6","lab6","la6","sib6","si6",
  "do7","reb7","re7","mib7","mi0","fa7","solb7","sol7","lab7","la7","sib7","si7",
  "do8","reb8","re8","mib8","mi8","fa8","solb8","sol8","lab8","la8","sib8","si8",
  "do9","reb9","re9","mib9","mi9","fa9","solb9","sol9","lab9","la9","sib9","si9",
 "do10","reb10","re10","mib10","mi10","fa10","solb10","sol10","lab10","la10","sib10"
,"si10"]!!(toInt indiceReal)

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

canalReal :: Real -> {#Char}
canalReal canal = ["canal 1","canal 2","canal 3","canal 4","canal 5",
                   "canal 6","canal 7","canal 8","canal 9","canal 10",
                   "canal 11","canal 12","canal 13","canal 14",
                   "canal 15","canal 16"]!!((toInt canal)-144)

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
listaDeInstrumentosMIDI :: Real -> {#Char}
listaDeInstrumentosMIDI indice =
    ["AcousticGrandPiano","BrightAcousticPiano",
    "ElectricGrandPiano","Honky-tonkPiano","ElectricPiano1",
    "ElectricPiano2","Harpsichord","Clavi","Celesta",
    "Glockenspiel","MusicBox","Vibraphone","Marimba",
```

```
"Xylophone","TubularBells","Dulcimer","DrawbarOrgan",
"PercussiveOrgan","RockOrgan","ChurchOrgan","ReedOrgan",
"Accordion","Harmonica","TangoAccordion",
"AcousticGuitar(nylon) ","AcousticGuitar(steel) ",
"ElectricGuitar(jazz) ","ElectricGuitar(clean) ",
"ElectricGuitar(muted) ","OverdrivenGuitar",
"DistortionGuitar","Guitarharmonics","AcousticBass",
"ElectricBass(finger)","ElectricBass(pick) ","FretlessBass",
"SlapBass1","SlapBass2","SynthBass1","SynthBass2","Violin",
"Viola","Cello","Contrabass","TremoloStrings",
"PizzicatoStrings","OrchestralHarp","Timpani",
"StringEnsemble1","StringEnsemble2","SynthStrings1",
"SynthStrings2","ChoirAahs","VoiceOohs","SynthVoice",
"OrchestraHit","Trumpet","Trombone","Tuba","MutedTrumpet",
"FrenchHorn","BrassSection","SynthBrass1","SynthBrass",
"SopranoSax","AltoSax","TenorSax","BaritoneSax","Oboe",
"EnglishHorn","Bassoon","Clarinet","Piccolo","Flute",
"Recorder","PanFlute","BlownBottle","Shakuhachi","Whistle",
"Ocarina","Lead1(square)","Lead2(sawtooth)","Lead3(calliope)",
"Lead4(chiff)","Lead5(charang)","Lead6(voice)","Lead7(fifths)",
"Lead8(bass+lead)","Pad1(newage)","Pad2(warm)","Pad3(polysynth)",
"Pad4(choir)","Pad5(bowed)","Pad6(metallic)","Pad7(halo)",
"Pad8(sweep) ","FX1(rain) ","FX2(soundtrack) ","FX3(crystal) ",
"FX4(atmosphere) ","FX5(brightness) ","FX6(goblins) ",
"FX7(echoes) ","FX8(sci-fi) ","Sitar","Banjo","Shamisen","Koto",
"Kalimba","Bagpipe","Fiddle","Shanai","TinkleBell","Agogo",
"SteelDrums","Woodblock","TaikoDrum","MelodicTom","SynthDrum",
"ReverseCymbal","GuitarFretNoise","BreathNoise","Seashore",
"BirdTweet","TelephoneRing","Helicopter","Applause",
"Gunshot"]!!(toInt indice)
```

```
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
//esta função converte uma lista de bytes de deltaTime para ppq

deltaTime :: [Int]-> Int
deltaTime novaListaTracksMtrksSemCabecalho

//          DELTATIME COM 1 BYTE

|primeiroByte < 128 = primeiroByte // deltaTime4 com 1 byte

//          DELTA TIME COM DOIS BYTES

|( primeiroByte > 127 &&
  segundoByte  < 128  ) =
  (primeiroByte bitand 64)/64 *2^13+
  (primeiroByte bitand 32)/32 *2^12+
  (primeiroByte bitand 16)/16 *2^11+
  (primeiroByte bitand 8)/8   *2^10+
  (primeiroByte bitand 4)/4   *2^9 +
  (primeiroByte bitand 2)/2   *2^8 +
  (primeiroByte bitand 1)/1   *2^7 +

  (segundoByte bitand 64)/64 *2^6 +
  (segundoByte bitand 32)/32 *2^5 +
  (segundoByte bitand 16)/16 *2^4 +
  (segundoByte bitand 8)/8 *2^3 +
  (segundoByte bitand 4)/4*2^2 +
  (segundoByte bitand 2)/2*2^1 +
  (segundoByte bitand 1)/1*2^0
```

```
//      DELTATIME COM TRÊS BYTES

|( primeiroByte > 127 &&
   segundoByte  > 127 &&
   terceiroByte < 128 ) =

   (primeiroByte bitand 64)/64 *2^20 +
   (primeiroByte bitand 32)/32 *2^19 +
   (primeiroByte bitand 16)/16 *2^18 +
   (primeiroByte bitand 8)/8 *2^17 +
   (primeiroByte bitand 4)/4*2^16 +
   (primeiroByte bitand 2)/2*2^15 +
   (primeiroByte bitand 1)/1*2^14 +

   (segundoByte bitand 64)/64 *2^13 +
   (segundoByte bitand 32)/32 *2^12 +
   (segundoByte bitand 16)/16 *2^11 +
   (segundoByte bitand 8)/8 *2^10 +
   (segundoByte bitand 4)/4*2^9  +
   (segundoByte bitand 2)/2*2^8  +
   (segundoByte bitand 1)/1*2^7  +

   (terceiroByte bitand 64)/64 *2^6 +
   (terceiroByte bitand 32)/32 *2^5 +
   (terceiroByte bitand 16)/16 *2^4 +
   (terceiroByte bitand 8)/8 *2^3 +
   (terceiroByte bitand 4)/4*2^2 +
   (terceiroByte bitand 2)/2*2^1 +
   (terceiroByte bitand 1)/1*2^0
```

```
//          DELTATIME COM QUATRO BYTES
|( primeiroByte > 127 &&
   segundoByte  > 127 &&
   terceiroByte > 127 &&
   quartoByte <128 ) =

   (primeiroByte bitand 64)/64 *2^27 +
   (primeiroByte bitand 32)/32 *2^26 +
   (primeiroByte bitand 16)/16 *2^25 +
   (primeiroByte bitand 8)/8 *2^24 +
   (primeiroByte bitand 4)/4*2^23 +
   (primeiroByte bitand 2)/2*2^22 +
   (primeiroByte bitand 1)/1*2^21 +

   (segundoByte bitand 64)/64 *2^20 +
   (segundoByte bitand 32)/32 *2^19 +
   (segundoByte bitand 16)/16 *2^18 +
   (segundoByte bitand 8)/8 *2^17 +
   (segundoByte bitand 4)/4*2^16+
   (segundoByte bitand 2)/2*2^15 +
   (segundoByte bitand 1)/1*2^14 +

   (terceiroByte bitand 64)/64 *2^13 +
   (terceiroByte bitand 32)/32 *2^12 +
   (terceiroByte bitand 16)/16 *2^11 +
   (terceiroByte bitand 8)/8 *2^10 +
   (terceiroByte bitand 4)/4*2^9+
   (terceiroByte bitand 2)/2*2^8 +
   (terceiroByte bitand 1)/1*2^7 +
   (quartoByte bitand 64)/64 *2^6 +
```

```
    (quartoByte bitand 32)/32 *2^5 +
    (quartoByte bitand 16)/16 *2^4 +
    (quartoByte bitand 8)/8 *2^3 +
    (quartoByte bitand 4)/4*2^2+
    (quartoByte bitand 2)/2*2^1 +
    (quartoByte bitand 1)/1*2^0

//        MENSAGEM DE ERRO PARA DELTATIME > 4 BYTES
|otherwise = abort "Nao existe deltaTime com mais de quarto bytes!"
//        CONSTANTES UTILIZADAS

where
    primeiroByte =: novaListaTracksMtrksSemCabecalho!!0
    segundoByte  =: novaListaTracksMtrksSemCabecalho!!1
    terceiroByte =: novaListaTracksMtrksSemCabecalho!!2
    quartoByte   =: novaListaTracksMtrksSemCabecalho!!3

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//MOSTRA TEXTUALMENTE A TONALIDADE DA MÚSICA

mostraTonalidade:: [[Int]]->String
mostraTonalidade x=:[[acidentes,tonalidade]]
|lista == [0,0]="Do maior"
|lista == [0,1]="la menor"
|lista == [1,0]="Sol maior"
|lista == [1,1]="Mi menor"
|lista == [2,0]="RE maior"
|lista == [2,1]="Si menor"
|lista == [3,0]="La maior"
|lista == [3,1]="Fa# menor"
```

```
|lista == [4,0]="Mi maior"
|lista == [4,1]="Do# menor"
|lista == [5,0]="Si maior"
|lista == [5,1]="SoL# menor"
|lista == [6,0]="Fa# maior"
|lista == [6,1]="Re# menor"
|lista == [7,0]="Do# maior"
|lista == [7,1]="La# menor"

|lista == [255,0]="Fa maior"
|lista == [255,1]="Re menor"
|lista == [254,0]="Sib maior"
|lista == [254,1]="Sol menor"
|lista == [253,0]="MIb maior"
|lista == [253,1]="Do menor"
|lista == [252,0]="Lab maior"
|lista == [252,1]="Fa menor"
|lista == [251,0]="Reb maior"
|lista == [251,1]="Sib menor"
|lista == [250,0]="Solb maior"
|lista == [250,1]="Mib menor"
|lista == [249,0]="Dob maior"
|lista == [249,1]="Ab menor"
 |lista == [256,256]= "*Do maior*"
 |otherwise = ("Tonalidade desconhecida!"+++
            "\n MetaEvento  "+++ "[" +++ (toString acidentes)+++","
            +++(toString tonalidade)+++"]"+++"  nao catalogado")

where lista =: hd x
//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
/* Função que calcula a tonalidade e o modo para bemóis. Quando for bemol(>127), o
    resultado do acidente vem multiplicado por 10 e a dezena informa o número de
    bemóis. Se for sustenido segue normalmente. /*

calculaAcidentes :: [Int]-> [Int]
calculaAcidentes tonalidadeParaCalculo=:[acidente,modo]
|acidente > 127 = [(256 - acidente)*10,modo]//maior que 127 -> bemol
|otherwise = tonalidadeParaCalculo

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

NBytesDeltaTime :: [Int] -> Int
NBytesDeltaTime lista
|  (primeiroByte < 128)  = 1 // UM BYTE
|( (primeiroByte > 127) &&
   (segundoByte  < 128) )= 2 // DOIS BYTES
|( (primeiroByte > 127) &&
   (segundoByte  > 127) &&
   (terceiroByte < 128) )= 3 // TRÊS BYTES
|otherwise = 4 // quatro bytes
where
 primeiroByte =: lista!!0
 segundoByte  =: lista!!1
 terceiroByte =: lista!!2

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
//pega elementos de uma lista

Primeiro :: [Int]-> Int
Primeiro lista = lista!!0
```

```
next :: [Int]-> Int
next lista  = lista!!1

next2 :: [Int]-> Int
next2 lista = lista!!2

next3 :: [Int]-> Int
next3 lista = lista!!3

next4 :: [Int]-> Int
next4 lista = lista!!4
```

A.2. MIDIBiblioteca.dcl

```
definition module MIDIBiblioteca
import StdEnv

//Batera x :: Int->{#Char}

listaDeInstrumentosMIDI :: Real -> {#Char}
tempoStringParaReal :: {#Char} -> Real
calculaTempo :: Real ->{#Char}
tempoReal :: Real -> {#Char}
volumeReal :: Real -> {#Char}
canalReal :: Real -> {#Char}
notaCifraRealBemol :: Real -> {#Char}
notaCifraRealSus :: Real -> {#Char}
ppqToDeltaTime :: Int -> [Int]
deltaTime :: [Int]-> Int
mostraTonalidade:: [[Int]]->String
```

```
calculaAcidentes :: [Int]-> [Int]
NBytesDeltaTime :: [Int] -> Int
Primeiro :: [Int]-> Int
next :: [Int]-> Int
next2 :: [Int]-> Int
next3 :: [Int]-> Int
next4 :: [Int]-> Int
```

A.3. Sistemas de Numeração - Uma Breve Abordagem

Para facilitar o entendimento do padrão MIDI e os Standard MIDI Files, é necessário conhecer os sistemas de numeração utilizados pelos processadores e computadores. O ser humano trabalha com o sistema de numeração decimal. O computador trabalha com o sistema de numeração binário. A literatura trata as mensagens MIDI no sistema de numeração hexadecimal. Sendo assim, este anexo se preocupa em fornecer as devidas informações para tal entendimento do padrão MIDI.

A.3.1. Conversão entre o Sistema Decimal e Outros Sistemas de Numeração

Para converter da base 10 para a base 2 e da base dois para a base 10 é relativamente simples. Observe: A lei de formação dos números na base dois segue o mesmo raciocínio utilizado na base 10, ou seja:

A.3.1.1 Base 10

O dígito menos significativo (da unidade) possui o seu valor (de **0** a **9**) multiplicado por $\mathbf{10^0}$ (1), o da casa das dezenas possui seu valor multiplicado por $\mathbf{10^1}$(10), o da casa das centenas multiplicado por $\mathbf{10^2}$ (100) e assim por diante, cada vez aumentando o expoente da base **10**.

Exemplo:

Dado o número 9453_{10}, escrevê-lo em uma tabela na forma de potências de **10**.

Resposta: $9453_{10}= 9 \times \mathbf{10^3} + 4 \times \mathbf{10^2} + 5 \times \mathbf{10^1} + 3 \times \mathbf{10^0}$. Colocando 9453_{10} em forma de tabela, temos:

$\times 10^3$	$\times 10^2$	$\times 10^1$	$\times 10^0$
9	4	5	3

A.3.1.2 Base 2

O bit menos significativo possui o seu valor (**0** ou **1**) multiplicado por $\mathbf{2^0}$ (1). O próximo possui seu valor multiplicado por $\mathbf{2^1}$ (2) e assim por diante, cada vez aumentando o expoente da base **2**.

Exemplo 1:

Dado o número 1010_2 escrevê-lo em uma tabela em forma de potências de **2** e determinar seu valor na base **10**.

Colocando 1010_2 em forma de tabela, tem-se:

$\times 2^3$ (=8)	$\times 2^2$ (=4)	$\times 2^1$ (=2)	$\times 2^0$ (=1)
1	0	1	0

Convertendo para a base **10**, tem-se que: $\mathbf{1010_2}= 1 \times \mathbf{2^3} + 0 \times \mathbf{2^2} + 1 \times \mathbf{2^1} + 0 \times \mathbf{2^0} = \mathbf{10_{10}}$.

Exemplo 2:

Dado o número 1110_2,escrevê-lo em uma tabela em forma de potências de **2** e determinar seu valor na base **10**.

Colocando 1110_2 em forma de tabela, tem-se:

$\times 2^3$ (=8)	$\times 2^2$ (=4)	$\times 2^1$ (=2)	$\times 2^0$ (=1)
1	1	1	0

Convertendo para a base **10**, tem-se que: $\mathbf{1110_2}= 1 \times \mathbf{2^3}+ 1 \times \mathbf{2^2}+ 1 \times \mathbf{2^1}+ 0 \times 2^0 = 8+4+2+0 = \mathbf{14_{10}}$

A.3.1.3 Base N

O número menos significativo possui o seu valor (de **0** a **N-1**) multiplicado por N^0, o próximo possui seu valor multiplicado por N^1 e assim por diante, cada vez aumentando o expoente da base **N**.

Exemplo:

Dado o número $N_3N_2N_1N_0$, convertê-lo para decimal e colocá-lo em forma de tabela de potências de **N**. N_3, N_2, N_1 e N_0 são números da base **N**. Assim, tem-se que: $N_3N_2N_1N_0 = N_3 \times N^3 + N_2 \times N^2 + N_1 \times N^1 + N_0 \times N^0 = ?_{10}$. O valor de **?** depende da base **N**. Esta é uma forma simples de converter qualquer base em decimal.

Colocando $N_3N_2N_1N_0$ em forma de tabela, tem-se:

x N^3	x N^2	x N^1	x N^0
N3	N2	N1	N0

A.3.1.4 Base 16

Hexadecimal - símbolos decimais mais seis símbolos do alfabeto. Base dezesseis, **N = 16** (dígitos de **0** a **9** + caracteres de **A** a **F**). Como só conhecemos dez símbolos para representar números, ao utilizar a base 16 encontramos o problema de ter que complementar símbolos a estes dez já existentes. Daí surgiu o sistema **hexadecimal**, ou seja, os dez símbolos do sistema decimal (de **0** a **9**), acrescentando seis símbolos do alfabeto (de **A** a **F**). Assim, temos que: **A = 10; B=11; C=12; D=13; E=14** e **F=15.**

Exemplo:

Dado o número $7D4_{16}$, convertê-lo em decimal temos que: $\mathbf{7D4_{16}} = 7\text{x}\mathbf{16^2} + D\text{x}\mathbf{16^1} + 4\text{x}\mathbf{16^0} = 7\text{x}\mathbf{256} + 13\text{x}\mathbf{16} + 4\text{x}\mathbf{1} = \mathbf{1830_{10}}$,

Colocando $7D4_{16}$ em forma de tabela, tem-se:

x 16^3	x 16^2	x 16^1	x 16^0
0	7	13 (**13=D**)	4

Referências Bibliográficas

[1] AMARAL, Mª Cristina Linhares Fonseca do. *O Computador no Processo Ensino – Aprendizagem.* Revista Tecnologia Educacional, R. J. v.7, n.61, nov. – dez. 1984.

[2] BENNETT, R. *Elementos Básicos da Música.* Rio de Janeiro: Jorge Zahar, 1990.

[3] BOOM, Michael. *Music Through MIDI: Using MIDI to create your own electronic music system.* Microsoft PRESS, Washington, 1987.

[4] CHEDIAK, Almir. *Dicionário de Acordes Cifrados.* 2. ed. Rio de Janeiro: Editora Irmãos Vitale S/A, 1984.

[5] CHEDIAK, Almir. *Harmonia e Improvisação.* Rio de Janeiro: Lumiar Editora, v. 1, 1986.

[6] COSTA, Clarissa L. da. *Uma Breve História da Música Ocidental.* São Paulo: Ars Poética, 1992.

[7] COSTA, Glinzer S. C. da Silva. *O Projeto de Informatização da Sala de Aula – Considerações Fundamentais.* Revista Tecnologia Educacional, R. J. v.25, n.128, jan. – fev. 1996.

[8] D'IPOLITO, Cláudio. *Desenvolvimento do Software Educacional: Arte exclusiva de informatas ou ao alcance do educador?* Revista Tecnologia Educacional, v16, (77): Jul./Ago. 1987.

[9] *Enciclopédia da Música Brasileira Erudita, Folclórica e Popular.* São Paulo: Art Ed., v.1, 1977.

[10] GUROVITZ, Helio. *Computação imita o Cérebro.* Folha de São Paulo, 28 de agosto, caderno 6 p. 16, 1994.

[11] HOWE, Hubert S. Jr.. *Creativity in Computer Music.* Byte, July, p. 158-173, 1979.

[12] IFEACHOR, Emmanuel C., JERVIS, Barrie W. *Digital signal processing - A practical aproach.* UK: Addison-Wesley Publishing Company Inc,1993.

[13] KOELLREUTTER, H. J. *Harmonia Funcional: Introdução à teoria das Funções Harmônicas*.Rio de Janeiro: Editora Ricordi, 1980. 71p.

[14] KUGEL, Peter. *Myhill's Thesis: There's More Than Computing in Musical Thinking*. Computer Music Journal, v.14, n.3, Fall, p.12-24, 1990.

[15] LACERDA, O. *Compêndio de Teoria Elementar da Música*. São Paulo: Ricordi Brasileira, 1961.

[16] LANGDELL, James. *The PC's New Frontier Music*. PC Magazine, April 29, p.187-201, 1986.

[17] LEFFA, Vilson J. *O Uso do computador na Produção de Material Didático*. Revista Tecnologia Educacional, R. J. v.7, n.77, jul. – ago. 1987.

[18] LEVACOV, Marília. *Avaliação de Software Educacional*. Revista Tecnologia Educacional, R. J. v.16, n.75/76, mar. – jun. 1987.

[19] LIMA, Luciano Vieira. *Desenvolvimento de um sistema de composição musical automática dirigida por estilo* (Tese de Doutorado). São Paulo: USP, 1998.

[20] LIMA, L. V., CARRIJO, G. A., DOVICCHI, J. C. *Music Based Coefficients for Wavelet Transform: A Contribuition to Musical Signals Analysys* In: AES - Audio Enginnering Society - 108th Convention 2000, 2000. AES - Audio Enginnering Society - 108th Convention 2000. , 2000.

[21] LIMA, L. V., et al. *Automatic Identification of Frequencies of Musical Notes in Polyphonics Wave Files* In: SBC&M 2000, 2000.

[22] LIMA, L. V., et al. *Editing Standard MIDI File in Functional Language CLEAN* In: Diderot99, 1999, Vienna, 1999.

[23] LIMA, L. V., FAVARO, V. V. *Conceitos básicos e estruturas de programação. Linguagem Funcional CLEAN* (Apostila). 2001.

[24] _____. *Binários e Booleanos, Operações Matemáticas, Inteiros longos, Vetores e Matrizes. Linguagem Funcional CLEAN* (Apostila). 2001.

[25] _____. Strings, Caracteres, Tuplas, registros e tipos abstratos de dados. *Linguagem Funcional CLEAN* (Apostila). 2001.

[26] LIMA, L. V., FAVARO, V. V., LOPES, G. F. Interfaces gráficas e estruturas de I/O. *Linguagem Funcional CLEAN* (Apostila). 2001.

[27] LIMA, L. V., MACHADO, A. C, CAMARGO JR., H. *Computer Music. Band-in-a-Box 10.0.* Playmusic. São Paulo: , v.44, p.7 - 9, 2001.

[28] LIMA, L. V., MACHADO, A. C., OLIVEIRA, D. C. *Computer Music -Sound Forge 5.0.* Playmusic. São Paulo: , v.46, p.7 - 9, 2001.

[29] LOY, Gareth, ABBOT, Curtis. *Programing Languages for Computer Music Systhesis, Performance and Composition.* Computer Surveys, v.17, n.2, june, p.235-245, 1985.

[30] MACHADO, A. C., et al. *Computação Musical: Encore 4.2.1 & Band-in-a-Box 10: Arranjo, Seqüenciamento e Editoração de Partituras.* São Paulo: Érica, 2001.

[31] MACHADO, A. C. LIMA, L. V. *Edição de Partituras por Computador - Encore Parte 1.* Playmusic. Brasil: , v.41, p.72 - 73, 2001.

[32] _____. *Edição de Partituras por Computador - Encore parte II.* Playmusic. São Paulo: , v.42, p.12 - 13, 2001.

[33] _____. *Edição de Partituras por Computador - Encore parte III.* Playmusic. São Paulo: , v.43, p.12 - 13, 2001.

[34] _____. *Edição de Partituras por computador - Encore parte IV.* Playmusic. São Paulo: , v.44, p.69 - 70, 2001.

[35] _____. *Edição de Partituras por computador - Encore parte V.* Playmusic. São Paulo: , v.45, p.78 - 79, 2001.

[36] _____. *Edição de Partituras por computador - Finale2001 parte I.* Playmusic. São Paulo: , v.46, p.14 - 16, 2001.

[37] _____. *Edição de Partituras por computador - Finale2001 parte II.* Playmusic. São Paulo: , v.47, p.12 - 13, 2001.

[38] _____. *Edição de Partituras por computador - Finale2001 parte III.* Playmusic. São Paulo: , v.48, p.14 - 15, 2001.

[39] MACHADO, A. C., LIMA, L. V., PINTO, M. M. *Computação Musical: Finale 2001: Arranjo e Editoração de Partituras.* São Paulo: Érica, 2001.

[40] _____. *Computação Musical: Cakewalk 9: Arranjo, Seqüenciamento e Editoração de Partituras*. São Paulo: Érica, 2001.

[41] MARINHO, Simão Pedro P. *Micro Computadores na Educação. Conclusões e recomendações de um Simpósio Internacional*. Revista Tecnologia Educacional, v16 (78/79), Set./Dez. 1987.

[42] MED, Bohumil. *Teoria da Música*. Brasília: Thesaurus Editora e Sistemas Áudio Visuais Ltda, 1980.

[43] *MIDI Detailed Specification 1.0*. International MIDI Association, 1983, 1991, 1994.

[44] MOOG, Robert A. *The Soundchaser Computer Music Systems*. Byte, Decembrer, p. 260-277, 1982.

[45] MOOG, Robert A. *Digital Music Synthesis*. Byte, June, p. 155-168, 1986.

[46] MOORE, F. Richard. *Música Computacional - Uma Breve Descrição*.

[47] MOORE, F. Richard. *Uma Abordagem tecnológica para a música*.

[48] MORAES, J. Jota de. *Música da Modernidade: origens da música do nosso tempo*. São Paulo: Editora Brasiliense, 1983.191p.

[49] OLIVEIRA, Celina Couto de Oliveira, MENEZES, Eliane Inez M., MOREIRA, Mércia. *Avaliação de Software Educativo*. Revista Tecnologia Educacional, R. J. v.16, n.77, jul. – ago. 1987.

[50] POWELL, Roger, GREAN, Richard. *Four MIDI Interfaces: MIDI interfaces for the Commodore 64, IBM PC, Macintosh, and Apple II family*. Hardware Review, June, p.265-272, 1986.

[51] PROAKIS, J. G., MANOLAKIS, D. G. *Introduction to Digital Signal Processing*. New York: Macmillian Publ. Co., 1988.

[52] RATTON, Miguel. *Criação de Música e Sons no Computador*. Rio de Janeiro: Editora Campus, 1995.

[53] _____. *MIDI – Guia Básico de Referência*.Rio de Janeiro: Editora Campus, 1992.

[54] _____. *Guia Rápido para o Band-in-a-Box*. Rio de Janeiro: Informus, 1998.

[55] _____. *Integração de MIDI e áudio no computador PC.* Resumo da palestra apresentada no 1° Workshop Música & Tecnologia, PUC-RJ, 15 de janeiro, 1996 .

[56] _____. MIDI e Copyright. Música & Tecnologia n.° 66, 1997.

[57] _____. *O que é Computer Music.* Revista Backstage, 1998.

[58] _____. *As Novas Técnicas Para se Fazer Música.* Revista Música & Tecnologia n.° 52, 1998.

[59] _____. *Áudio na Internet - Sonorizando homepages.* 1997.

[60] ROADS, C., STRAWN, J.(EDS). *Foundations of Computer Music.* Cambridge, Mass.: MIT Press, 1988.

[61] ROADS, Curtis. *Research in Music and Artificial Intelligence.* Computing Surveys, v. 17, n.2, June, p.163-190, 1985.

[62] STEHMAN, Jacques. *História da Música Européia.* 2. ed. Enciclopédia de Bolso Bertrand, 1979.

[63] TRAGTENBERG, Livio. *Contraponto: Uma Arte de Compor.* São Paulo: Editora da Universidade de São Paulo, 1994.