UNIVERSIDADE DE SÃO PAULO

Escola de Artes, Ciências e Humanidades

Relatório Técnico PPgSI-003/2021

Análise técnica da arquitetura i3-2310M 64 Bits e sua comparação com a arquitetura AMD FX-4100 Quad-Core

Otto Alves Antonioli
Raphael Janofsky Pivato
Norton Trevisan Roman

Junho - 2021

O conteúdo do presente relatório é de única responsabilidade dos autores.

Série de Relatórios Técnicos

PPgSI-EACH-USP
Rua Arlindo Béttio, 1000 – Ermelino Matarazzo
03828-000 – São Paulo, SP.
TEL: (11) 3091-8197 http://www.each.usp.br/ppgsi

Análise técnica da arquitetura i3-2310M 64 Bits e sua comparação com a arquitetura AMD FX-4100 Quad-Core

Otto Alves Antonioli[1] **, Raphael Janofsky Pivato**[2] **, Norton Trevisan Roman**[3]

[1]Escola de Artes, Ciências e Humanidades – Universidade de São Paulo
São Paulo – SP, Brazil

ottoalves22@usp.br

[2]Escola de Artes, Ciências e Humanidades – Universidade de São Paulo
São Paulo – SP, Brazil

raphaeljpivato@usp.br

[3]Escola de Artes, Ciências e Humanidades – Universidade de São Paulo
São Paulo – SP, Brazil

nortontr@gmail.com

Resumo. *Este documento discorre sobre a arquitetura i3-2310M 64 Bits, analisando-a desde seu contexto histórico, prosseguindo pelas motivações que culminaram em seu surgimento e desenvolvimento, seu uso atual, seu tipo, o hardware escolhido, a configuração dos componentes, seu desempenho de acordo com o benchmark oficial, o comportamento estabelecido pelo conjunto de instruções utilizado em sua confecção; concluindo comparando-a com a arquitetura AMD FX-4100 Quad-Core, de modo a estabelecer os aspectos positivos e negativos explorados por cada uma delas, sob a perspectiva de cada item supracitado.*

Palavras-chave: arquitetura de computadores, i3-2310M 64 Bits, AMD FX-4100 Quad-Core

1. Objetivo

Este documento tem por objetivo analisar a arquitetura i3-2310M 64 Bits, visando ao fornecimento de informações claras e concisas - quando dentro do possível - a seu respeito, proporcionando uma fonte embasada para consultas e comparações futuras.

Sendo a qualidade da análise parte intrínseca do objetivo deste trabalho, cabe aqui explicitar seus principais aspectos. O foco dos autores deste documento foi em principalmente obter dados diretamente dos canais de informação disponibilizados pelo fabricante da arquitetura selecionada. Sendo assim, as fontes consultadas provêm do meio digital em sua maioria, em formato de livros, manuais oficiais, páginas da web e relatórios técnicos; tendo em vista a coleta de informações necessárias para fornecer insumos para a viabilização das medições de desempenho e de configuração das instruções da arquitetura; caracterizando pesquisa de cunho bibliográfico. [Silveira e Gerhardt 2009].

Primeiramente é realizada a apresentação inicial da arquitetura, contendo seu contexto histórico, como a geração pertencente, alguns aspectos em que a arquitetura difere de suas anteriores, e as motivações que levaram à sua comercialização. A arquitetura é caracterizada, tendo seus componentes descritos e então seguindo para uma análise dos dados

obtidos com medições de seu desempenho. Para tal medição, é também realizada uma comparação com a arquitetura AMD FX-4100 Quad-Core.

Feita a comparação, conclui-se exaltando os principais aspectos positivos e negativos de cada arquitetura, visando ao fornecimento de informações consolidadas no que concerne à análise de arquiteturas modernas e suas comparações.

2. Histórico

Esta seção é detalhada em duas partes. Uma caracteriza o contexto histórico e seu uso atual, que em suma correspondem a como os chips foram inicialmente fabricados, como evoluíram cronologicamente, culminando na elaboração da arquitetura escolhida para análise neste relatório, até o momento presente, em que estruturas mais modernas são comercializadas. A segunda parte introduz em aspecto mais técnico a arquitetura, pondo em pauta qual era o modelo mais avançado tecnologicamente em momento anterior à sua elaboração, para elucidar quais as motivações para sua criação (e posterior comercialização), bem como as diferenças entre elas e também as vantagens que a arquitetura descrita dispõe sobre sua predecessora.

2.1. Contexto histórico e uso atual

Fundada em 1968 pelos especialistas em semicondutores Gordon Moore e Robert Noyce, a Intel® se consolidou no mercado sendo uma das principais fabricantes globais de microprocessadores. Em 1971, o 4004 foi desenvolvido, sendo o primeiro microchip fabricado na história com velocidade inicial de clock de 108KHz, 2300 transistores e litografia[1] de 10 micron. [Tecmundo® 2018].

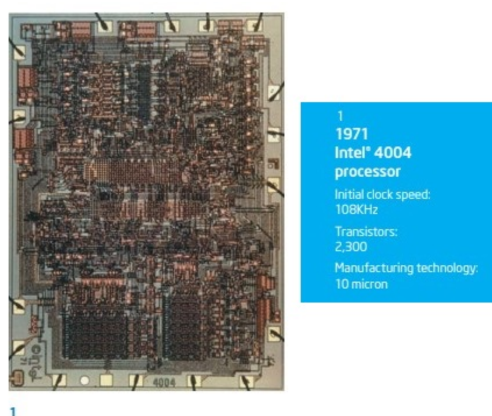

Figura 1. Processador Intel 4004. [Intel® 2012]

A arquitetura i3-2310M 64 Bits é de fabricação da Intel e pertence à família de processadores Intel Core 2ª Geração, caracterizada pela velocidade inicial de clock de 3.8GHz, 1.16 bilhões de transistores e litografia de 32 nanômetros. [Intel® 2019b].

[1]Litografia refere-se à tecnologia de semicondutor usada para fabricar um circuito integrado e é (atualmente) expressa em nanômetro (nm), que indica o tamanho dos recursos integrados ao semicondutor. [Intel® 2019a]

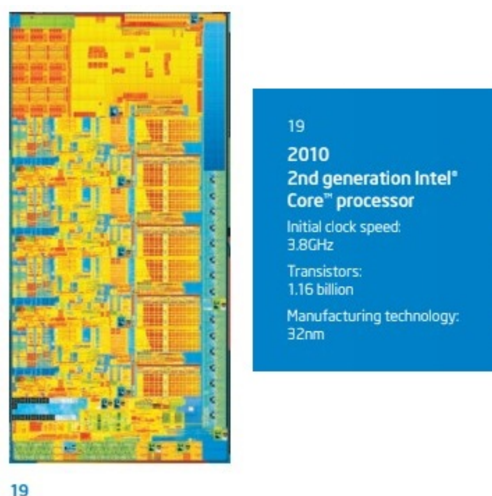

Figura 2. Processador Intel Core 2ª Geração. [Intel® 2012]

A evolução dos chips ao longo do tempo é disposta na Figura 3, exemplificando visualmente algumas consequências previstas pela Lei de Moore (o próprio Gordon Moore, fundador da Intel), cuja previsão é amplamente difundida nos meios acadêmicos e corporativos e que estipulara que o adensamento do número de transistores em um único chip dobraria a cada dois anos, aumentando o poder de computação disponível e possibilitando melhor aproveitamento de espaço.

Na Figura 3 é possível verificar como os transistores dos chips ficaram mais densos pelo fato de as linhas de contorno que os delimitam iniciarem bem definidas e, à medida em que evoluem (da esquerda para a direita, de cima para baixo), os traços vão se tornando cada vez mais agrupados até aparentarem ser borrões dentro dos chips. Também é possível visualizar redução de tamanho da placa, o que permitiu a produção de chips pequenos o suficiente para serem utilizados em dispositivos, como laptops e celuares, que além de menores eram mais poderosos que seus antecessores.

A família de processadores Intel Core 2ª Geração inclui os chips i3, i5 e i7, possuindo o codinome *Sandy Bridge* - por motivos tratados na subseção seguinte - e são encontrados majoritariamente em computadores pessoais e notebooks. Contudo, sua fabricação fora descontinuada, visto que novas gerações foram desenvolvidas e a substituíram no mercado. [Intel® 2019a].

2.2. Motivação para o uso da arquitetura

O codinome *Sandy Bridge* foi dado a essa família de processadores por realizar uma melhoria de construção na microarquitetura dos chips em relação ao seu predecessor, conforme ilustrado na Figura 4. O modelo anterior - *Nehalem*, também conhecido pelos codinomes *Arrandale* ou *Clarkdale* - consistia em dois chips, sendo um deles os componentes da CPU[2] e o outro os componentes da GPU[3], conectados por QPIs[4] formando

[2]Central Processing Unit - Unidade de Processamento Central, consistindo nos Cores - núcleos - do processador e da LLC (ou também L3) - Last Level Cache ou Large Level 3 Cache.

[3]Graphics Processing Unit - Unidade de Processamento Gráfico.

[4]Quick Path Interconnect - Interconexão de Trajeto Rápido

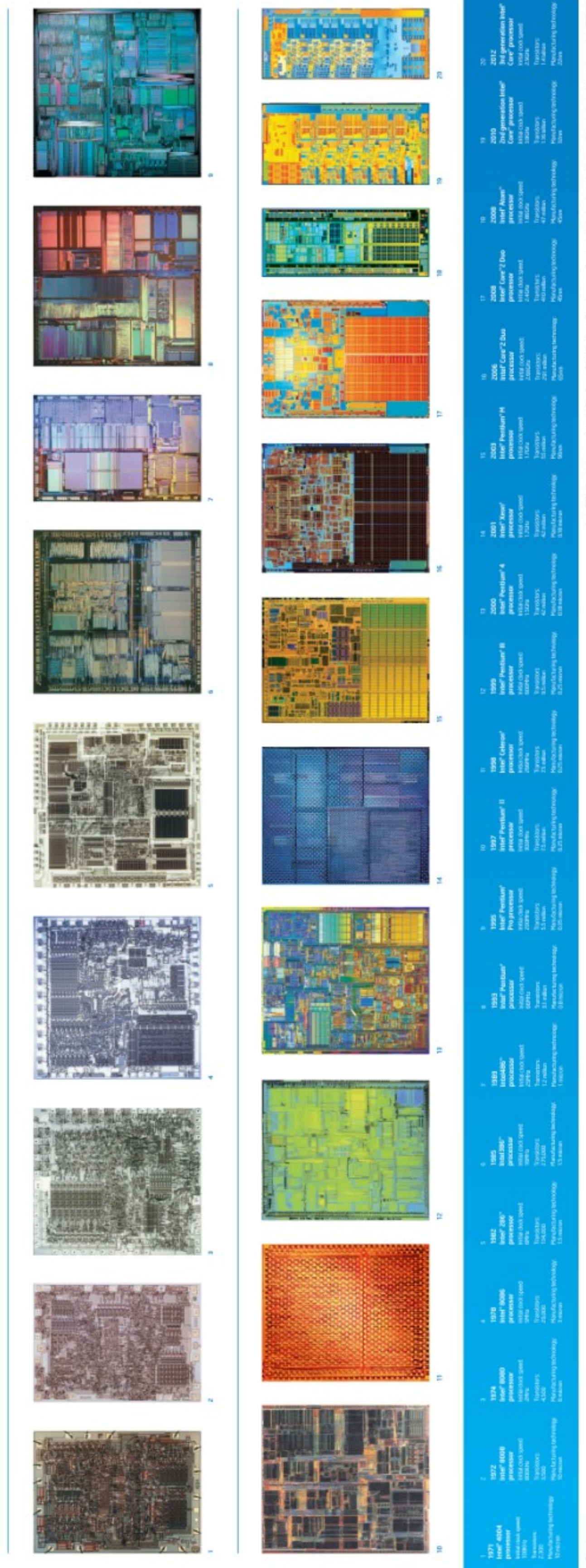

Figura 3. Linha do tempo evolutiva dos chips Intel. [Intel® 2012]

uma ponte entre os chips. Ambos os chips eram então agrupados em um mesmo pacote. Já na *Sandy Bridge*, os conectores dos chips foram dispensados e a CPU e a GPU foram agrupadas em um mesmo chip, ou seja, a ponte que antes existia para interconectar tais componentes agora consistia na própria matéria prima do chip, o silício - componente primário da areia (*sand*). [Gwennap 2010].

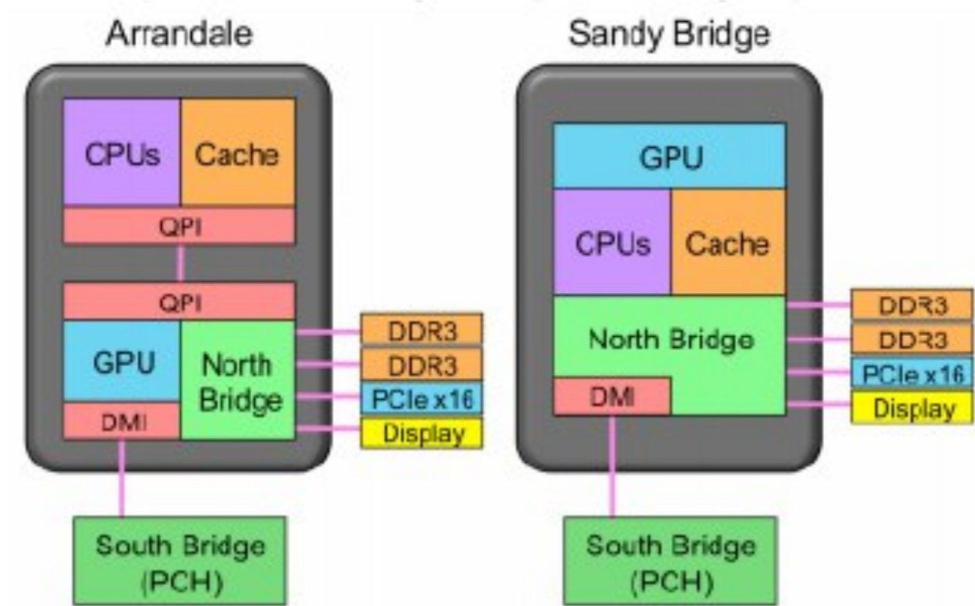

Figura 4. Comparação da integração da GPU nos chips da Primeira Geração Intel com a Segunda Geração Intel. [Gwennap 2010]

A *Sandy Bridge* apresentou, em relação ao modelo predecessor, melhorias de implementação microestrutural no preditor de *branches*, no renomeamento de registradores e na decodificação de instruções. Além da eliminação de um chip externo, a proximidade entre CPU e GPU promove um modesto ganho de performance em velocidade máxima, mas, principalmente, um aumento de eficiência em seu consumo energético e de ganhos no modo turbo de funcionamento. Tal implementação reduz o consumo de bateria, proporcionando vida útil extendida no caso dos notebooks, pois o completamento de tarefas mais rápido permite ao sistema retornar o quanto antes para estado ocioso - de baixo consumo. Comercialmente, o público-alvo dessa evolução foram aplicações multimídia - som, imagem e vídeo - e de processamento gráfico em 3D. [Gwennap 2010].

O posicionamento estratégico da GPU no chip integrado permite que ela tenha acesso à LLC[5], encurtando o caminho para a memória principal, aumentando a velocidade do processamento gráfico e reduzindo a sobrecarga sobre a banda do barramento da DRAM. Esse novo design da GPU, entre outros detalhes[6], culminou em uma aproximação aos modelos amplamente usados por outros fabricantes de GPUs externas, garantindo desempenho pouco inferior aos componentes mais avançados disponíveis no mercado, porém suficientemente bom para que, em aplicações de uso geral, torne-se dispensável a necessidade de dispositivos de processamento gráficos externos, acarretando no provimento de

[5]LLC ou também L3: Last Level Cache ou Large Level 3 Cache.

[6]Verificar relatório referenciado: SANDY BRIDGE SPANS GENERATIONS. [Gwennap 2010]

uma GPU mais barata e de consumo reduzido/otimizado. [Gwennap 2010].

3. Detalhes

No que concernem os detalhes técnicos que caracterizam a arquiteura i3-2310M 64 Bits, esta seção aborda inicialmente a *pipeline* projetada, destacando a estruturação de seus componentes e o fluxo de dados que as micro-operações, em nível genérico, percorrem. Em sequência, os registradores são listados e suas utilidades explanadas. Por fim, são destrinchadas as principais configurações de frequência base de *clock*, quantidade de núcleos, *threads*, memória e cache disponíveis.

3.1. Pipeline

A microarquitetura *Sandy Bridge* foi projetada visando à estruturação de uma *pipeline* seccionada em dois macro-estágios: um inicial, que trata de cada instrução lida em ordem, decodificando-as em micro-operações; e outro, de execução, cuja ordem das instruções independe da ordem em que foram lidas, sendo capaz de manter até seis micro-operações em execução por ciclo. [Intel® 2016].

Além desses estágios, a finalização é realizada pela *Retire Unit* - Unidade de Conclusão - garantindo que os resultados da execução das micro-operações, incluindo quaisquer exceções que possam ter sido disparadas, estejam visíveis de acordo com a ordem estabelecida pelo programa em execução. [Intel® 2016].

A Figura 5 ilustra os componentes organizacionais da *pipeline* de uma microarquitetura *Sandy Bridge* e a sequência do fluxo de dados pode ser sumarizada do seguinte modo [Intel® 2016]:

1. A unidade de leitura (*Fetch Unit*) obtém a próxima instrução do programa em execução;
2. A instrução é passada à BPU (*Branch Predictor Unit* - Unidade de Predição de Ramificações), capaz de eficientemente prever o alvo de uma ramificação, determinando o próximo bloco de código a ser executado e liberando a CPU para rodar as próximas instruções muito antes do estabelecimento do resultado da *branch*. Entre essas instruções encontram-se:
    - *Branches* condicionais;
    - Chamadas diretas a métodos e *jumps*;
    - Chamadas indiretas a métodos e *jumps*;
    - Retornos.

   O processador, então, irá procurar pelo próximo bloco de código, na seguinte ordem, nos respectivos componentes:
    (a) Na Cache de instruções decodificadas (L0);
    (b) Na Cache de instruções, ativando a *pipeline* de decodificação legada (L1);
    (c) Na Cache L2, na LLC e na memória principal, conforme a necessidade.
3. A micro-operação correspondente ao código obtido é enviada ao bloco de *Rename/Dispatch* - Renomear/Despachar - responsável por validar a micro-operação recém chegada.
   Em caso de exceções (*Faults, Traps*), estas são sinalizadas, enviadas à *Retire Unit* e prontamente busca-se a próxima micro-operação, de modo a garantir o máximo de eficiência e uso da estrutura da *pipeline*.

Caso sejam válidas, as micro-operações são despachadas ordenadamente ao *Scheduler* - escalonador (não presente nessa representação, mas situado em uma camada entre o despachante e os componentes coloridos em roxo: ALUs[7], armazenamento, carga, somador e multiplicador) - responsável por gerenciar a ordem de execução a depender do melhor fluxo de dados. Para micro-operações simultaneamente prontas, é utilizada uma estrutura de FIFO (*First In First Out* - primeiro a chegar é o primeiro a sair, ordem de chegada).

No caso de detecção de uma previsão errada por parte da BPU, há o redirecionamento direto da micro-operação resultante do caminho correto, caso em que o processador pode sobrepujar o trabalho realizado com o percurso anterior (errado), realizando o trabalho com o percurso correto, evitando assim a formação de bolhas na *pipeline*.

Instruções que usam da memória principal são gerenciadas e reordenadas, dentro do possível, de modo a atingir máximo paralelismo e performance, uma vez que costumam utilizar maior quantidade de estágios da *pipeline*;

4. Executada a instrução, seu resultado se encontrará em algum registrador ou na memória, a depender do seu tipo, fechando o ciclo e dando início a uma nova leitura.

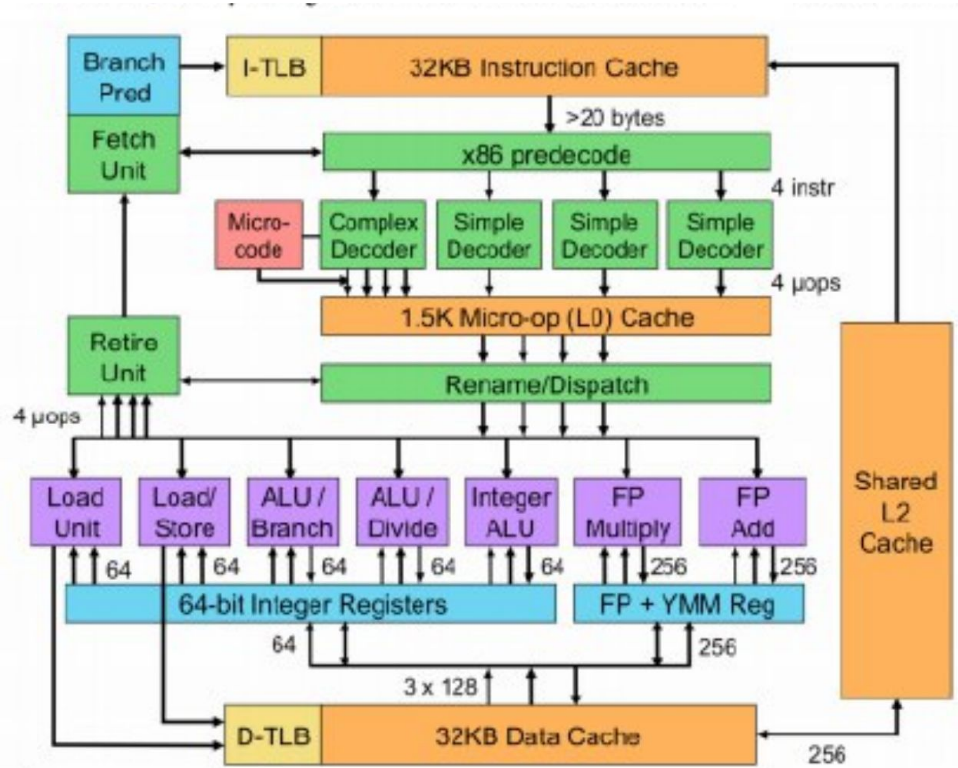

Figura 5. *Pipeline* da microarquitetura Sandy Bridge. [Gwennap 2010]

[7] *Arithmetic and Logic Unit* - Unidade de Lógica e Aritmética

3.2. Registradores

Registradores Gerais (*General-Purpose Registers*): Conforme ilustrado na Figura 6, existem 8 registradores de 32 bits para serem utilizados como operandos de operações lógicas, aritméticas e para cálculos de endereços ou ponteiros para regiões de memória, enumerados abaixo [Intel® 2016]:

- EAX - Acumulador de operandos e dados de resultados.
- EBX - Ponteiro para o *Data Segment*.
- ECX - Utilizado como contador em instruções de *strings e loops*.
- EDX - Ponteiro de Entrada/Saída.
- ESI - Utilizado como fonte em instruções de *strings* e ponteiro auxiliar para o registrador de *Data Segment DS*.
- EDI - Utilizado como ponteiro de destino em instruções de *strings* e destino ou ponteiro para segmentos do registrador ES do *Data Segment*.
- EBP - Ponteiro para dados na pilha.
- ESP - Este registrador em especial contém o ponteiro para o topo da pilha de processos, e convencionou-se não ser usado para qualquer outro propósito.

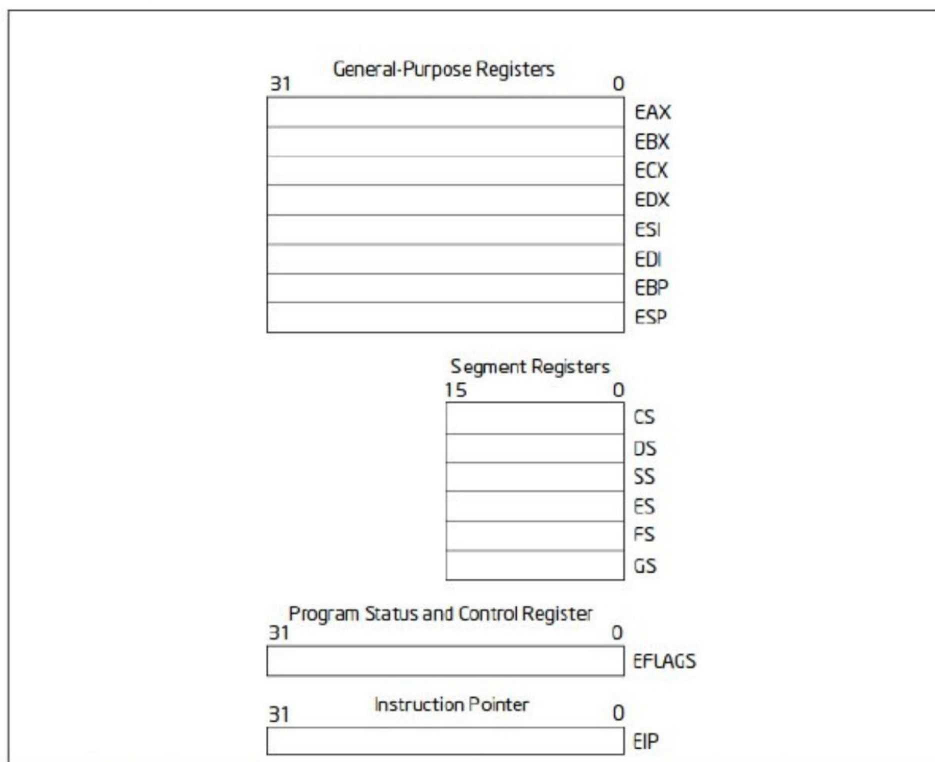

Figura 6. Registradores Gerais da micro-arquitetura *Sandy Bridge*. [Intel® 2019b]

Registradores de Segmento (*Segment Registers*): CS, DS, SS, ES, FS e GS. Estes registradores possuem seletores de segmento de 16 bits, que identificam um segmento de memória. Ao usar modelos de memória não-segmentada, cada registrador inicia no endereço zero, podendo chegar até 4 GB de segmentos lineares de memória, como visto na Figura 7 [Intel® 2016]:

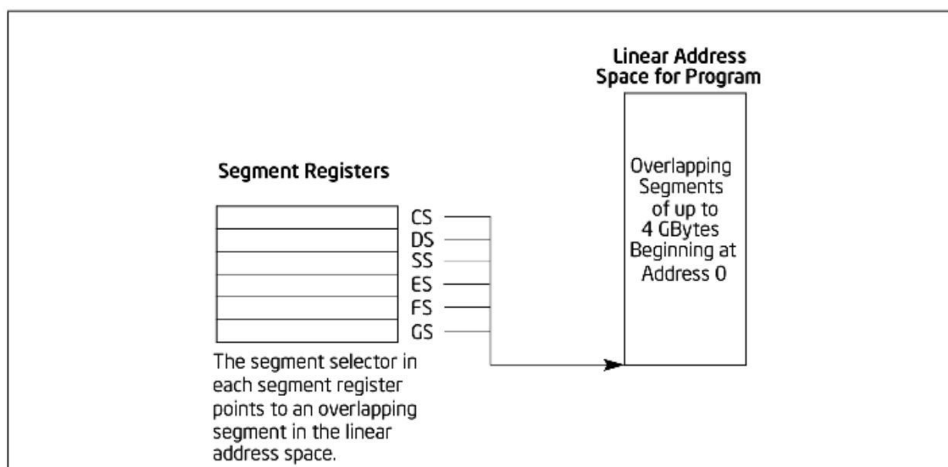

Figura 7. Registradores de *Data Segment* utilizando modelo *flat* (não-segmentado) de memória. [Intel® 2019b]

Utilizando memória segmentada, cada registrador é carregado, ordenadamente, tal qual em uma lista ligada, com um segmento que aponta para um endereço de memória, como na Figura 8. Para acessar um segmento não apontado por qualquer um destes, um seletor deve ser carregado para se tornar acessível por um registrador de segmento. [Intel® 2016].

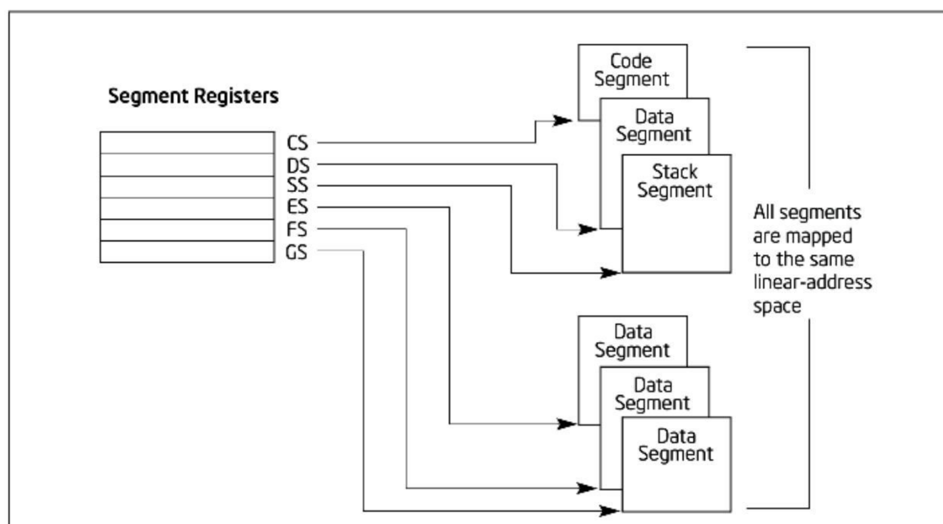

Figura 8. Registradores de *Data Segment* utilizando modelo segmentado de memória. [Intel® 2019b]

Cada registrador de segmento é associado a um tipo de armazenamento, entre código, dado ou pilha. Seletores de segmentos de código guardam as instruções que estão sendo executadas, que serão posteriormente buscadas pelo processador. Seletores de segmentos de dados guardam dados de diferentes módulos, compartilhados ou em estruturas alocadas dinamicamente, existindo quatro registradores que armazenam ponteiros para este tipo de segmento. O seletor de segmento de pilha, contido no registrador SS, tem a função de

armazenar a pilha de processos e é sempre utilizado por todas operações que envolvam a pilha de processos. Os registradores de segmento são divididos da seguinte maneira (como pode ser visto também na Figura 8) [Intel® 2016]:

- CS - Segmento de código
- DS - Segmento de dados
- SS - Segmento de pilha
- ES - Segmento de dados
- FS - Segmento de dados
- GS - Segmento de dados

Registradores EFLAGS: O conjunto de registradores EFLAGS contém um grupo de *flags* de status, uma *flag* de controle e um subconjunto de *flags* de sistema, como visto na Figura 9 [Intel® 2019b]:

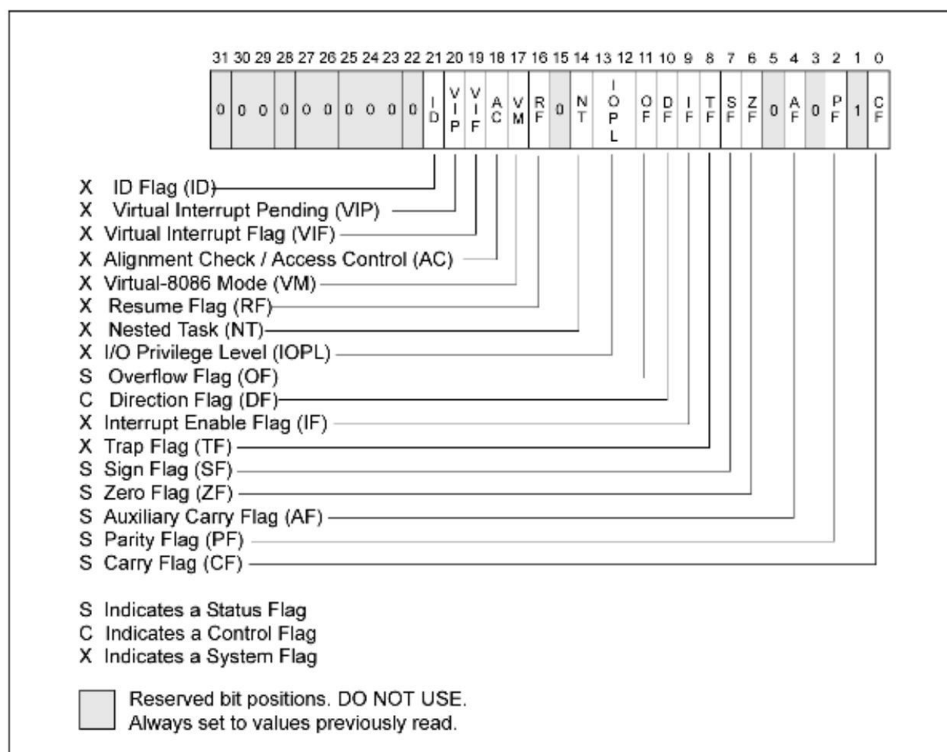

Figura 9. Registradores EFLAGS. [Intel® 2019b]

Alguns destes registradores podem ser transferidos com certas instruções para o registrador comum EAX ou para a pilha de processos; e após transferidas, as *flags* podem então serem acessadas ou modificadas com as instruções BT, BTS, BTR e BTC. [Intel® 2019b].

Quando um processo é interrompido, o processador grava o estado atual das EFLAGS no segmento *Task State* (segmento do estado da tarefa) da memória e, ao carregar um processo que já havia sido iniciado, lê as EFLAGS do mesmo segmento de memória. No caso de interrupção ou exceção, os registradores EFLAGS também serão salvos na pilha de processos, e no caso de uma troca de processo, salvos no segmento *Task State*. [Intel® 2019b].

As *Flags* dos registradores EFLAGS, ilustradas na Figura 9, operam conforme descrição abaixo:

- CF - *Carry Flag*, bit 0, indica se uma operação aritmética ”vai um” dos bits mais significantes do resultado, além de indicar overflow; se não, inativo (0).
- PF - *Parity Flag*, bit 2, ativo caso o byte menos significativo do resultado contenha um número ímpar de bits 1.
- AF - *Auxiliary Carry Flag*, bit 4, ativo caso uma operação aritmética ”vai um” no bit 3 do resultado; se não, inativo (0).
- ZF - *Zero Flag*, bit 6, ativo se o resultado for zero; se não, inativo (0).
- SF - *Sign Flag*, bit 7, terá o mesmo valor do bit menos significativo do resultado aritmético (0 para valores maiores que zero, 1 para valores menores que zero).
- OF - *Overflow Flag*, bit 11, ativo caso um resultado aritmético seja maior que um valor positivo válido ou menor que um valor negativo válido; se não, inativo (0).
- DF - *Direction Flag*, controla instruções de *strings*. Ao estar ativo, auto-decrementa tais instruções, para percorrer *strings* do maior endereço para o menor. Inativo, o mesmo incrementa as mesmas, percorrendo-as no sentido contrário.

Flags de Sistema: Controlam chamadas ao sistema operacional. São estes [Intel® 2019b]:

- TF - *Trap Flag*, bit 8, ao ser ativado, liga o modo passo a passo.
- IF - *Interrupt Flag*, bit 9, controla a resposta do processador para chamadas de interrupção, estando ativo e inativo.
- IOPL - *I/O Privilege level field*, bits 12 e 13, indica o nível de privilégio do processo atual.
- NT - *Nested Task Flag*, bit 14, controla o fluxo de chamada de processos interrompidos. Ativo quando o processo atual é relacionado com o último processo executado; caso contrário, inativo.
- RF - *Resume Flag*, bit 16, controla respostas do processador para excessões de *debug*.
- VM - *Virtual-8086 mode Flag*, bit 17, ativa o modo 8086. Inativo, retorna ao modo protegido sem a semântica de 8086.
- AC - *Alignment Check Flag*, bit 18, checa se o modo usuário é o modo vigente, ativando-se neste caso, concedendo modo supervisor, se ativo.
- VIF - *Virtual Interrupt Flag*, bit 19, imagem virtual do registrador IF, usado em conjunto com o registrador VIP.
- VIP - *Virtual Interrupt Pending Flag*, bit 20, ativo para indicar uma interrupção pendente.
- ID - *Identification Flag*, bit 21, indica o uso da instrução CPIUID.

Ponteiro de instruções: Guarda o segmento de código da próxima instrução a ser executada. Este registrador não é acessível diretamente, porém pode ser acessado por meio de uma instrução CALL, que o lê e retorna o ponteiro de instrução da pilha de processos. [Intel® 2016].

3.3. Dados gerais do processador

O modelo i3-2310M, da arquitetura *Sandy Bridge*, possui frequência baseada em processador (ou frequência de *clock*) de 3.10GH, além de dispor de 2 núcleos com 4 *threads* cada, totalizando 8 *threads* neste modelo de chip, conforme Tabela 1. [Intel® 2019a].

Tabela 1. Tabela de Desempenho i3-2310M. [Intel® 2019a]

Desempenho	
Número de núcleos	2
Nº de threads	4
Frequência baseada em processador	3.10 GHz
Cache	3 MB Intel® Smart Cache
Velocidade do barramento	5 GT/s
TDP	65 W

Quanto aos atributos de memória, possui cache de 3MB da categoria Intel Smart Cache, com tamanho máximo de memória de 32GB, aceitando os tipos DDR3 1066 e DDR3 1333, com largura de banda máxima de 21GB por segundo, como descrito na Tabela 2. [Intel® 2019a].

Tabela 2. Especificações de memória i3-2310M. [Intel® 2019a]

Especificações de memória	
Tamanho máximo de memória (de acordo com o tipo de memória)	32 GB
Tipos de memória	DDR3 1066/1333
Nº máximo de canais de memória	2
Largura de banda máxima da memória	21 GB/s

4. Conjunto de instruções

No que concernem as informações contidas nesta seção, bem como as imagens utilizadas para ilustrar o formato e a estrutura das instruções, todas foram obtidas por meio do Intel® 64 and IA-32 Architectures Software Developer's Manual, fornecido pelo fabricante da arquitetura ([Intel® 2019b]).

4.1. Sintaxe

A estrutura sintática das instruções utilizadas pelo Assembly da microarquitetura *Sandy Bridge* corresponde a subconjuntos compreendidos no seguinte formato:

label: mnemônico argumento1,argumento2,argumento3

Em que:

- *label* - rótulo - é um identificador sucedido por dois pontos, sendo o seu uso obrigatório a depender da instrução executada, facultativo nos casos de uso geral;
- menmônico é uma palavra reservada que representa uma classe de *opcodes* - códigos de operação - que possuem igual função;
- argumento1, 2 e 3 são operandos opcionais, limitados de zero a três, a depender do *opcode* empregado. Os argumentos podem variar entre constantes (estabelecidas pelos usuários), registradores (sendo estes palavras reservadas, definidas pela estrutura de registradores utilizada pela microarquitetura) e *labels* (sendo estes palavras reservadas ou definidos pelos usuários).

Quando ao menos dois argumentos são utilizados para uma instrução lógica ou aritmética, o(s) operando(a) à direita é(são) o(s) fonte(s), enquanto o da esquerda é o de destino. [Intel® 2019b].

4.2. Formato

As instruções da arquitetura i3-2310M 64 Bits não possuem tamanho fixo, pois seu intuito é possuir uma organização parametrizável a depender da instrução, flexibilizando seu uso durante a codificação dos programas e *softwares* que operam sobre ela. Portanto, o formato é único, mas a obrigatoriedade do uso de seus elementos integrantes é variável.

Outro ponto digno de nota é o fato de o fabricante ter demonstrado cuidado quando da evolução de uma arquitetura para a próxima, no sentido de que os conjuntos de instruções atualizados possuem organizações complexas de modo a serem capazes de executar as instruções legadas de seus predecessores de modo transparente aos programadores de sistemas; além de disponibilizarem bits ou até mesmo bytes ainda sem uso - ou de uso raríssimo - com a perspectiva de que novas versões eventualmente surjam e requeiram tal uso.

A Figura 10 descreve o formato das instruções da arquitetura i3-2310M 64 Bits.

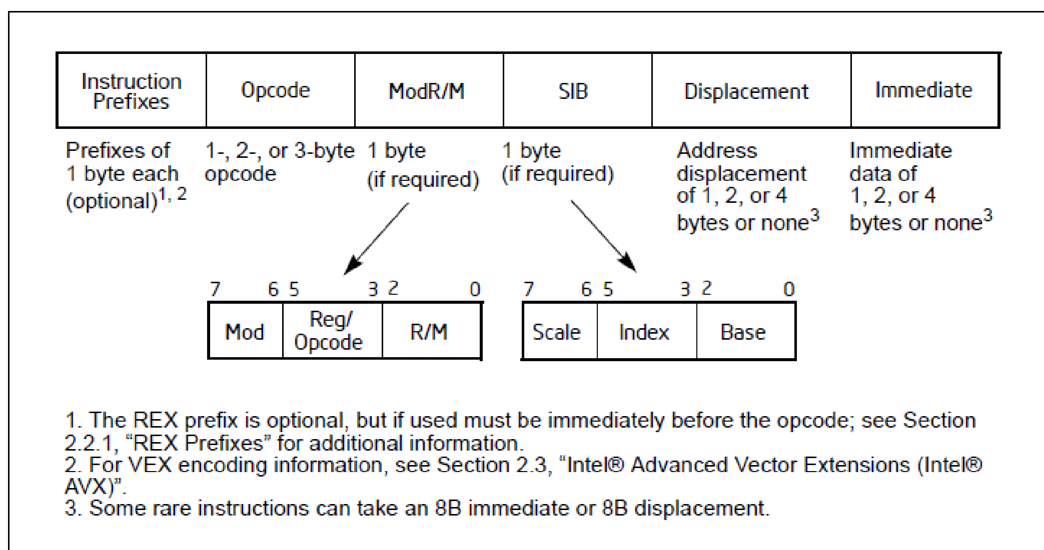

Figura 10. Formato das instruções.[Intel® 2019b]

Em que pode-se sumarizar:

- **Prefixos das Instruções:** divididos em 4 grupos (de trava e repetição, de predição de *branches* e sobreposição de segmentos, de sobreposição do tamanho de operandos; e de sobreposição do tamanho de endereços), sendo seu uso facultativo e limitado a um máximo de 4 prefixos (um por grupo), cada qual ocupando 1 byte em memória, independentes da ordem. Este campo também é utilizado para identificação de instruções legadas, desse modo promovendo um diferente arranjo para os bytes dos demais campos contidos em uma instrução;
- ***Opcodes*****:** correspondendo ao identificador da instrução tratada, tendo comprimento de 1 a 3 bytes e, para alguns casos específicos, o campo de 3 bits adicionais da seção ModR/M (seguinte) pode ser acionado;
- **ModR/M:** tendo comprimento de 1 byte e sendo utilizado por instruções que acessam a memória. Os campos Mod e R/M podem ser combinados originando 32 valores, sendo 8 registradores e 24 modos de endereçamento (o campo Reg/*Opcode* pode especificar um registrador ou completar um *opcode*). O campo R/M pode especificar um registrador como operando ou realizar a combinação com o campo Mod supracitada;
- **SIB:** o *Scale-Index-Base* - Escala-Índice-Base - também possui 1 byte de comprimento, é segmentado em três campos (SIB) e é utilizado como campo para endereçamento adicional do ModR/M, de 32 bits, em algumas instruções. Escala é um fator escalar, Índice é o número do registrador índice, e Base é o número do registrador base;
- ***Displacement*****:** o campo de deslocamento corresponde a outra forma de endereçamento, podendo ter comprimento de 1, 2 ou 4 bytes;
- ***Immediate*****:** o campo de operando imediato sempre acompanha o comprimento utilizado pelo de deslocamento, podendo também possuir 1, 2 ou 4 bytes, dependendo da instrução.

4.3. Principais instruções e sua organização no *Data Path*

As principais instruções da arquitetura i3-2310M 64 Bits estão listadas nesta subseção e sua sintaxe segue a especificação descrita na subseção 4.1.

- **ADD:** responsável por realizar a soma (*add*) de inteiros com ou sem sinais do primeiro operando (destino) e o segundo (fonte), atribuindo o resultado ao primeiro:

  Lógica funcional: DESTINO = DESTINO + FONTE
  Exemplo de uso: ADD AX,15

  Em que ADD é o mnemônico (*opcode* da instrução), AX é o argumento 1 (operando 1, que armazenará o resultado da soma - destino), e 15 é o argumento 2 (operando 2, cujo valor imediato, expresso em decimal, é 15 - fonte). Portanto, executada a instrução, AX conterá o valor que possuía antes da execução acrescido em 15 unidades (AX = AX + 15).
  Os operandos podem ser registradores, valores imediatos (somente o fonte) ou locais de memória (porém não dois locais de memória simultaneamente). A codificação da operação ADD é apresentada na Figura 11;

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA
MR	ModRM:r/m (r, w)	ModRM:reg (r)	NA	NA
MI	ModRM:r/m (r, w)	imm8/16/32	NA	NA
I	AL/AX/EAX/RAX	imm8/16/32	NA	NA

Figura 11. Codificação de uma instrução ADD. [Intel® 2019b]

- **ADDPD:** responsável por realizar a soma de dois, quatro ou oito valores compactados de ponto flutuante de dupla precisão (*add packed double-precision floating-point values*), do primeiro operando fonte com o segundo, armazenando o resultado no operando de destino. Como o armazenamento de um número em ponto flutuante de dupla precisão em memória é variável, o comportamento do deslocamento dos dados também acompanham tais nuances, ou seja, os operandos podem ser registradores de diferentes usos comuns, locais em memória de diferentes tamanhos ou até mesmo espaços vetoriais em memória, tornando a interpretação da instrução um tanto mais complexa que um simples ADD:

  Lógica funcional:DESTINO = FONTE1 + FONTE2
  Exemplo de uso: ADDPD [0000],EAX,[0064]

  Em que ADDPD é o mnemônico (*opcode* da instrução), [0000] é o argumento 1 (operando 1, que armazenará o resultado da soma, sendo a área de memória cujo endereço no segmento de dados é 0000 - destino), EAX é o argumento 2 (operando 2, sendo o registrador que contém previamente um valor em ponto flutuante e que será usado como argumento na soma - fonte 1), e [0064] é o argumento 3 (operando 3, sendo a área de memória cujo endereço no segmento de dados é 0064 e que será usada como argumento na soma - fonte 2). Portanto, executada a instrução, [0000] conterá o valor da soma entre EAX e [0064] ([0000] = EAX + [0064]).
  Essa forma é conceitualmente mais didática, servindo para a compreensão de seu funcionamento em linhas gerais, conforme apresentado na Figura 12;

Instruction Operand Encoding

Op/En	Tuple Type	Operand 1	Operand 2	Operand 3	Operand 4
A	NA	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA
B	NA	ModRM:reg (w)	VEX.vvvv	ModRM:r/m (r)	NA
C	Full	ModRM:reg (w)	EVEX.vvvv	ModRM:r/m (r)	NA

Figura 12. Codificação de uma instrução ADDPD.[Intel® 2019b]

- **AND e ANDN:** responsáveis por realizar uma comparação *bitwise* - bit a bit - da operação 'E' (*and*) ou 'E' negado (*and not*) entre o primeiro operando (de destino) e o segundo (fonte), armazenando o resultado no primeiro. Os operandos podem ser registradores ou locais de memória (porém não dois locais de memória simultaneamente):

  Lógicas funcionaias: DESTINO = DESTINO E FONTE
  DESTINO = DESTINO E NÃO FONTE

Exemplos de uso: AND AX,CX
ANDN AX,CX

Em que AND e ANDN são os mnemônicos (*opcodes* da instrução), AX é o argumento 1 (operando 1, que armazenará o resultado da comparação - destino), e CX é o argumento 2 (operando 2 - fonte). Portanto, executada a instrução, AX conterá o valor das paridades, se AND, ou não paridades, se ANDN, *bitwise* entre AX e CX (AX = AX E CX ou AX = AX E NÃO CX).
A comparação é realizada do bit menos significativo para o mais significativo. Quando os bits comparados entre ambos os operandos for igual a um, gera-se um bit com um como resposta; caso contrário, zero.
AND e ANDN possuem mesma codificação que a instrução ADD, conforme Figura 11;

- **CMP:** é responsável por realizar uma comparação (*compare*) entre os operandos por meio da subtração do segundo operando pelo primeiro, atualizando as *flags* de status do registrador EFLAGS de acordo com o resultado:

  Lógica funcional: EFLAGS atualizadas de acordo com OP2 - OP1
  Exemplo de uso: CMP AX,10

  Em que CMP é o mnemônico (*opcode* da instrução), AX é o argumento 1 (operando 1 - op1), e 10 é o argumento 2 (operando 2, cujo valor imediato, expresso em decimal, é 10 - op2). Portanto, executada a instrução, as EFLAGS serão atualizadas de acordo com o resultado de 10 - AX. Assim, é possível verificar qual dos operandos é maior, menor ou se são iguais, sendo critérios bastante usuais em instruções que dependam de condicionais.
  Os operandos podem ser registradores, valores imediatos ou locais de memória (porém não dois locais de memória simultaneamente). A codificação dos operandos desta instrução pode ser vista na Figura 13;

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r)	ModRM:r/m (r)	NA	NA
MR	ModRM:r/m (r)	ModRM:reg (r)	NA	NA
MI	ModRM:r/m (r)	imm8/16/32	NA	NA
I	AL/AX/EAX/RAX (r)	imm8/16/32	NA	NA

Figura 13. Codificação de uma instrução CMP. [Intel® 2019b]

- **DEC e INC:** por serem unárias, utilizam somente um operando (destino), sendo que esse tem seu conteúdo decrementado ou incrementado em uma unidade. O operando pode ser um registrador ou um local de memória:

  Lógicas de uso: DESTINO = DESTINO - 1
  DESTINO = DESTINO + 1
  Exemplos de uso: DEC AX
  INC [0008]

Em que DEC e INC são os mnemônicos (*opcodes* da instrução), e AX e [0008] são o argumento 1 (operando 1 - destino). Portanto, executada a instrução DEC AX, AX terá seu valor decrementado em um (AX = AX - 1); e executada a instrução INC [0008], o conteúdo armazenado na área de memória cujo endereço é 0008 terá sido incrementado em um ([0008] = [0008] + 1).
Vale mencionar que a implementação dessas instruções não interfere em modificações na *flag* CF, sendo recomendáveis para parametrização de *loops*, por exemplo. A codificação das instruções DEC e INC é apresentada na Figura 14;

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r, w)	NA	NA	NA
O	opcode + rd (r, w)	NA	NA	NA

Figura 14. Codificação das instruções DEC e INC. [Intel® 2019b]

- **DIV:** realiza uma divisão sem sinal do primeiro operando (dividendo) pelo segundo operando (divisor), armazenando o resultado no primeiro:

  Lógica funcional: DIVIDENDO = DIVIDENDO / DIVISOR
  Exemplo de uso: DIV AX,CX

  Em que DIV é o mnemônico (*opcode* da instrução), AX é o argumento 1 (operando 1 - dividendo), e CX é o argumento 2 (operando 2 - divisor). Portanto, executada a instrução, a operação realizada será equivalente a AX/CX tendo o resultado atribuído em AX (AX = AX/CX).
  O resultado pode ser expresso pelo quociente e pelo resto, que ficam armazenados cada qual em uma metade do registrador que antes expressava apenas o dividendo. O acesso isolado a esses dados é sintaticamente determinado pela substituição do sufixo 'X' por 'H' (*high* - sendo a metade dos bits mais significativos do registrador) ou 'L' (*low* - sendo a metade dos bits menos significativos do registrador). Dessa forma, após a instrução ser executada, um acesso a AX conteria o resto concatenado ao quociente da divisão, a AH apenas o resto e a AL apenas o quociente. O operando dividendo pode ser um registrador ou um local de memória. Na Figura 15 é descrita a codificação da instrução e da ação de divisão;

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (w)	NA	NA	NA

Table 3-15. DIV Action

Operand Size	Dividend	Divisor	Quotient	Remainder	Maximum Quotient
Word/byte	AX	r/m8	AL	AH	255
Doubleword/word	DX:AX	r/m16	AX	DX	65,535
Quadword/doubleword	EDX:EAX	r/m32	EAX	EDX	$2^{32} – 1$
Doublequadword/ quadword	RDX:RAX	r/m64	RAX	RDX	$2^{64} – 1$

Figura 15. Codificação de uma instrução DIV. [Intel® 2019b]

- **Jcc:** é responsável por gerar uma ramificação na sequência do código, saltando para um endereço (ou rótulo) que esteja situado no mesmo segmento de código

que o Jcc (*jump if condition is met*), caso a condição especificada tenho sido validada após checagem das *flags* do registrador EFLAGS. Vale destacar que o Jcc é uma denominação genérica e que diversos *opcodes* podem ser categorizados sob essa nomenclatura, tais como JG (*jump if greater* - saltar se maior), JGE (*jump if greater or equal* - saltar se maior ou igual), JL (*jump if less* - saltar se menor), JLE (*jump if less or equal* - saltar se menor ou igual), JZ (*jump if zero* - saltar se igual a zero), JNZ (*jump if not zero* - saltar se não for igual a zero):

Lógica funcional: Jcc RÓTULO
Exemplo de uso: JZ LOOP1

Em que JZ é o mnemônico (*opcode* da instrução) de categoria Jcc e LOOP1 é o argumento 1 (operando 1 - rótulo que identifica uma determinada linha do código executado). Se a(s) *flag(s)* referente(s) ao comando executado atender(em) a condição da instrução, o salto ocorrerá para o rótulo passado e a cadência de execução continuará a partir da linha de código por ele rotularizada; caso contrário, a instrução executada será a imediatamente seguinte ao Jcc. No exemplo em questão, a *flag* ZF será analisada: Se estiver ativa (igual a um) o salto ocorrerá; senão, o fluxo de execução segue inalterado.
A codificação desta instrução pode ser vista na Figura 16.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
D	Offset	NA	NA	NA

Figura 16. Codificação de uma instrução Jcc. [Intel® 2019b]

- **JMP:** é responsável por transferir o programa para um novo fluxo de instruções (*jump* - salto) a partir da especificação de um endereçamento, podendo esse ser um valor imediato, um registrador ou um local de memória, rotularizados conforme especificação do programador:

  Lógica funcional: JMP RÓTULO
  Exemplo de uso: JMP METODO2

  Em que JMP é o mnemônico (*opcode* da instrução) e METODO2 é o argumento 1 (operando 1 - rótulo que identifica uma determinada linha do código executado). Executada a instrução, o salto ocorrerá para o rótulo passado e a cadência de execução continuará a partir da linha de código por ele rotularizada.
  A codificação da instrução JMP pode ser vista na Figura 17;

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
D	Offset	NA	NA	NA
M	ModRM:r/m (r)	NA	NA	NA

Figura 17. Codificação de uma instrução JMP. [Intel® 2019b]

- **MOV:** é responsável por copiar (*move* - mover) o conteúdo do segundo operando (fonte) para o primeiro (destino). Os operandos podem ser registradores ou locais de memória. O fonte também pode ser um valor imediato:

  Lógica funcional: DESTINO = FONTE
  Exemplo de uso: MOV AX,3

  Em que MOV é o mnemônico (*opcode* da instrução), AX é o argumento 1 (operando 1 - destino dos dados), e 3 o argumento 2 (operando 2 - fonte dos dados, que neste caso foi expresso pelo valor imediato em decimal equivalente a 3). Executada a instrução, AX estará armazenando o valor 3 (AX = 3). Vale destacar que o conteúdo prévio do registrador é perdido após a execução desta instrução.
  Os operandos e a codificação da instrução podem ser vistos na Figura 18;

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MR	ModRM:r/m (w)	ModRM:reg (r)	NA	NA
RM	ModRM:reg (w)	ModRM:r/m (r)	NA	NA
FD	AL/AX/EAX/RAX	Moffs	NA	NA
TD	Moffs (w)	AL/AX/EAX/RAX	NA	NA
OI	opcode + rd (w)	imm8/16/32/64	NA	NA
MI	ModRM:r/m (w)	imm8/16/32/64	NA	NA

Figura 18. Codificação de uma instrução MOV. [Intel® 2019b]

- **MUL:** realiza uma multiplicação sem sinal do primeiro operando (destino) com o segundo (fonte), armazenando o resultado no primeiro:

  Lógica funcional: DESTINO = DESTINO * FONTE
  Exemplos de uso: MUL AL,CL
  MUL AX,CX
  MUL EAX,ECX

  Em que MUL é o mnemônico (*opcode* da instrução), AL, AX e EAX são o argumento 1 (operando 1 - destinos), e CL, CX e ECX são o argumento 2 (operando 2 - fontes).
  Como a probabilidade de *overflow* na multiplicação é maior que nas demais operações, existe previsão de diferentes formas de armazenamento a depender do tamanho dos operandos utilizados. O produto de dois números em formato binário de mesmo tamanho possui, no máximo, tamanho equivalente ao dobro dos dois termos da multiplicação, como 1111 * 1111 = 1110 0001 (primeiro número com 4 bits multiplicado ao segundo, também com 4 bits, resulta em um número com 8 bits) ou 11111111 * 11111111 = 1111 1110 0000 0001 (primeiro número com 8 bits multiplicado ao segundo, também com 8 bits, resulta em um número com 16 bits). Assim, a depender dos registradores utilizados como operandos, mais de um registrador será necessário para armazenamento do resutado. Para ilustrar esse comportamento, foram utilizados 3 exemplos de uso para esta instrução.
  Para MUL AL,CL; o armazenamento seria realizado em AX (AX = AL * CL). Para MUL AX,CX; o armazenamento seria realizado no par DX:AX (DX:AX

= AX * CX). E para MUL EAX,ECX; o armazenamento seria realizado no par EDX:EAX (EDX:EAX = EAX * ECX).
O primeiro operando deve ser um registrador (podendo ser composto, a depender do tamanho - atuando como um espelho da divisão), enquanto o fonte pode ser um registrador ou um local de memória. A codificação da instrução MUL é apresentada na Figura 19;

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r)	NA	NA	NA

Table 4-9. MUL Results

Operand Size	Source 1	Source 2	Destination
Byte	AL	r/m8	AX
Word	AX	r/m16	DX:AX
Doubleword	EAX	r/m32	EDX:EAX
Quadword	RAX	r/m64	RDX:RAX

Figura 19. Codificação de uma instrução MUL. [Intel® 2019b]

- **NOT:** é uma operação unária, na qual o operando passado - podendo ser um registrador ou um local de memória - é negado (*not* ou *one's complement negation* - negação do complemento de um), ou seja, cada um se torna zero e cada zero se torna um:

  Lógica funcional: DESTINO = NOT DESTINO
  Exemplo de uso: NOT [0000]

  Em que NOT é o mnemônico (*opcode* da instrução) e [0000] é o argumento 1 (operando 1 sendo a área de memória cujo endereço é 0000 - destino), que armazenará como resultado seu complemento *bitwise*.
  A codificação da instrução NOT é apresentada na Figura 20;

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M	ModRM:r/m (r, w)	NA	NA	NA

Figura 20. Codificação de uma instrução NOT. [Intel® 2019b]

- **OR:** é responsável por realizar um OU (*or*) inclusivo *bitwise* entre o primeiro operando (destino) e o segundo (fonte), armazenando o resultado no primeiro. A comparação é realizada do bit menos significativo para o mais significativo e cada comparação resulta em zero caso ambos sejam zerados; caso contrário, resulta em um:

  Lógica funcional: DESTINO = DESTINO OR FONTE
  Exemplo de uso: OR AX,CX

  Em que OR é o mnemônico (*opcode* da instrução), AX é o argumento 1 (operando 1, que armazenará o resultado da comparação - destino), e CX é o argumento 2

(operando 2 - fonte). Executada a instrução, AX conterá o valor das paridades regidas pela lógica do OU inclusivo *bitwise* entre AX e CX (AX = AX OU CX).
O operando fonte pode ser um valor imediato, um registrador ou um local de memória; enquanto o operando de destino é limitado a registradores e locais de memória. Contudo, não é permitido que ambos os operandos sejam locais de memória em uma mesma instrução.
OR possui mesma codificação que a instrução ADD, conforme Figura 11;

- **SHLD e SHRD:** sendo responsáveis por realizar deslocamentos lógicos nos bits do primeiro operando (destino) no passo estabelecido pelo segundo operando (contador). O deslocamento pode ser à esquerda (*double precision shift left*) ou à direita (*double precision shift right*):

  Lógica funcional: DESTINO = DESTINO deslocado em CONTADOR bits (à esquerda ou à direita)
  Exemplos de uso: SHLD AX,2
  SHRD AX,4

  Em que SHLD e SHRD são os mnemônicos (*opcodes* da instrução), AX é o argumento 1 (operando 1, que armazenará o resultado do deslocamento de bits - destino), e 2 ou 4 são o argumento 2 (operando 2, que contabiliza de maneira decimal a quantidade de bits que devem ser deslocadas - contador). Portanto, executada a instrução SHLD, AX conteria o valor que possuía antes tendo todos os seus bits deslocados à esquerda em 2 bits. Caso fosse executada a instrução SHLD, AX conteria o valor que possuía antes tendo todos os seus bits deslocados à direita em 4 bits.
  O operando destinatário pode ser um registrador ou um local em memória. Já o contador é limitado a valores imediatos (entre 0 e 63, dependendo do tamanho dos operandos utilizados) e ao registrador CL. A codificação desta instrução pode ser vista na Figura 21;

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
M1	ModRM:r/m (r, w)	1	NA	NA
MC	ModRM:r/m (r, w)	CL	NA	NA
MI	ModRM:r/m (r, w)	imm8	NA	NA

Figura 21. Codificação de instruções SHIFT. [Intel® 2019b]

- **SUB:** responsável por realizar subtração (*subtract*) do segundo operando (registrador fonte) pelo primeiro (registrador de destino), atribuindo o resultado no primeiro (subtração de inteiros com ou sem sinais):

  Lógica funcional: DESTINO = DESTINO - FONTE
  Exemplo de uso: SUB AX,6

  Em que SUB é o mnemônico (*opcode* da instrução), AX é o argumento 1 (operando 1, que armazenará o resultado da subtração - destino), e 6 é o argumento 2 (operando 2, representado pelo valor decimal imediato 6 - fonte). Executada

a instrução, AX conterá o valor de 6 subtraído do valor que previamente estava armazenado no registrador (AX = AX - 6).
Os operandos podem ser registradores, valores imediatos (somente o fonte) ou locais de memória (porém não dois locais de memória simultaneamente).
SUB possui mesma codificação que a instrução ADD, conforme Figura 11;

- **XOR:** é responsável por realizar um OU exclusivo *bitwise* entre o primeiro operando (destino) e o segundo (fonte), armazenando o resultado no primeiro. A comparação é realizada do bit menos significativo para o mais significativo e cada comparação resulta em zero caso sejam iguais, caso contrário resulta em um:

  Lógica funcional: DESTINO = DESTINO XOR FONTE
  Exemplo de uso: XOR AX,CX

  Em que XOR é o mnemônico (*opcode* da instrução), AX é o argumento 1 (operando 1, que armazenará o resultado da comparação - destino), e CX é o argumento 2 (operando 2 - fonte). Executada a instrução, AX conterá o valor das paridades regidas pela lógica do OU exclusivo *bitwise* entre AX e CX (AX = AX OU EXCLUSIVO CX).
  O operando fonte pode ser um valor imediato, um registrador ou um local de memória; enquanto o operando destinatário é limitado a registradores e locais de memória. Contudo, não é permitido que ambos os operandos sejam locais de memória em uma mesma instrução.
  XOR possui mesma codificação que a instrução ADD, conforme Figura 11.

5. Desempenho por *Benchmark*

Dado que as capacidades singulares de cada componente presente em uma arquitetura não são suficientes para comparar de modo assertivo seu desempenho, uma bateria de testes é realizada, visando a levar cada componente ao seu limite, além, principalmente, de verificar o comportamento que o todo apresenta quando na exigência do uso em conjunto de cada parte para o completamento de tarefas específicas. Com isso, a parametrização do comparativo é enriquecida e fornece uma melhor noção das capacidades que uma determinada CPU de fato possui. Os resultados desses testes são chamados de *benchmarks* e são expostos nesta seção.

Na carência de meios acadêmicos para que fossem produzidos dados certificados acerca do *benchmark* da arquitetura tratada neste documento, bem como em encontrar documentos fornecidos pelo fabricante, os autores do presente recorreram a dados fornecidos pela multinacional PassMark® Software Pty Ltd, líder de um grupo de desenvolvimento de *software* privado, especializada no fornecimento de soluções de alta performance fundamentadas em *benchmarks* e detentora de um dos maiores bancos de dados de *benchmarks* disponibilizados gratuitamente para a comunidade de usuários global.

Para a confecção desse banco de dados, são coletadas informações tanto provenientes de dados fornecidos por usuários quanto pelos trabalhos de pesquisa desenvolvidos pelas equipes integrantes do grupo, assim, é digno de menção que as informações aqui expostas possuem algum risco de não confiabilidade. [PassMark® 2019a].

5.1. Critérios de teste de desempenho

Para geração dos resultados, nove testes são realizados, sendo atribuído um *score* a cada um, em que *scores* mais altos significam uma melhor colocação. Os critérios levados em conta são os seguintes. [PassMark® 2021]:

1. **Teste de operações com números inteiros** - sendo um teste com um grande número de instruções básicas, de modo a verificar a taxa de transferência ”limpa” que a CPU possui, sendo a de melhor desempenho aquela que executar o teste em menor tempo;
2. **Teste de compressão** - utilizando estruturas de dados complexas e verificando a maior taxa de compressão que a CPU consegue manter sem que haja perda da integridade dos dados originais;
3. **Teste de números primos** - tendo seu resultado expresso em operações por segundo, visa à verificação do quão rapidamente uma CPU pode executar um algoritmo de validação de números primos que opera em *loops*;
4. **Teste de criptografia** - utiliza-se estruturas de diversos métodos de criptografia em blocos de dados aleatórios, de modo a que apenas quem detiver a chave de descriptação poderá realizar a engenharia reversa, novamente o tempo de completamento da tarefa sendo o principal apontador do desempenho da CPU;
5. **Teste de operações com ponto flutuante** - análogo ao teste com inteiros. Entretanto, pelo fato de o armazenamento e o comportamento de operações que fazem uso de ponto flutuante serem de complexidade mais elevada, o teste aponta o quão eficiente uma CPU é na obtenção de resultados de mais alta precisão;
6. **Teste de instruções estendidas** - utiliza os subconjuntos mais complexos de instruções (a depender da arquitetura suportá-las ou não), sendo elas FMA, AVX e SSE, de modo a que o resultado seja a média aritmética das três. Apesar de conceitualmente mais complexas, essas instruções foram projetadas para realizar algumas operações matemáticas mais rapidamente. Assim, o resultado desse teste é comparado com os outros testes matemáticos, verificando a eficiência que a CPU pode proporcionar;
7. **Teste de ordenação de *Strings*** - faz uso do algoritmo de *Quicksort* para ordenar *Strings* de 1 byte, sendo novamente o tempo para a conclusão da tarefa o parâmetro de medição;
8. **Teste físico** - faz uso da *Bullet Physics Engine* para medir, durante um dado intervalo de tempo, a quantidade de vezes que a CPU é capaz de repetir uma simulação de interação física. Maiores quantidades representam mais velocidade de processamento, sob a perspectiva desse teste;
9. **Teste de *Thread* única** - utiliza somente um núcleo lógico da CPU e o submete a testes complementares de ponto flutuante, ordenação e compressão, pois ainda hoje é comum encontrar aplicações que utilizam somente um *core*.

5.2. Medição realizada

Após submissão aos testes supracitados de 2078 amostras de máquinas com a arquitetura i3-2310M, foi gerado o *score* médio de 2432. Uma análise mais elaborada sobre a representatividade desse valor é feita na seção seguinte. [PassMark® 2019b].

6. Comparação

A comparação entre arquiteturas foi pensada como uma forma de aplicar as informações contidas neste relatório, de forma que não sejam apenas uma lista de componentes técnicos, demonstrando como benchmark e especificações poderiam ser utilizados por um usuário, possibilitando fazer uma escolha embasada entre processadores. Desta forma, foi de intuito dos autores optar por uma arquitetura lançada no mesmo ano da i3-2310M, mas de diferente fabricante, com componentes individualmente mais potentes para justamente verificar se haveria coincidência em um todo com melhor *score*/desempenho ou se poderia ser feito um apontamento contraditório. Sendo assim, foi escolhida a arquitetura AMD FX-4100 Quad-Core.

Ante o *score* de 2432 obtido pelo i3-2310M 64 bits, o AMD FX-4100 Quad-Core resultou em um *score* de 4076 como média após testes realizados em 2124 amostras, ou seja, teve um resultado consideravelmente superior, acompanhando - nesse caso - o esperado. [PassMark® 2019b]

Essa arquitetura se distingue da i3-2310M por ser da classe de CPU de Desktop, possuir frequência de *clock* de 3.6GHz e suportar *overclocking* de até 3.8GHz de turbo velocidade; o que, para os amplos testes realizados, de fato demonstrou ser um diferencial que justificasse seu desempenho mais elevado.

Contudo, há um preço para esse maior desempenho. A PassMark® também realiza outros apontamentos em *benchmark* que não somente se resumem ao desempenho das arquiteturas. Há também medições baseadas em custo de mercado, medição exclusivas em uma única *thread*, custo energético, entre outros parâmetros.

Nesses outros âmbitos, os *scores* do i3-2310M são mais competitivos e, em alguns casos, melhores que os do FX-4100, no que segue:

1. **Custo:** I3-2310M possui *score* de 62.68 enquanto FX-4100 26.57;
2. ***Single Thread Rating*:** I3-2310M possui *score* de 1,008 enquanto FX-4100 1,222.

O valor monetário de um i3-2310M é muito mais baixo, especialmente se for levada em consideração a aquisição de múltiplos chips.

O resultado das medições em uma única *thread*, em contrapartida, aproxima mais as duas arquiteturas, conotando diferenças sutis entre elas no que concernem o método de *pipeline* adotado e a configuração das instruções.

7. Conclusão

Analisar arquiteturas é uma atividade complexa, por existir um grande número de variáveis que interferem na produção de resultados. Também vale mencionar o quão relativa é a afirmação de que uma é melhor que outra, dado que podem ter sido projetadas com diferentes propósitos. Tomando como exemplo os resultados produzidos na elaboração deste documento, tais questões são evidenciadas.

Ao comparar uma arquitetura da classe desktop com uma de notebook há uma diferença de propósito nelas, que nada tem a ver com questões de desempenho, e sim, de mobilidade. Se a necessidade de um determinado usuário é ter uma máquina móvel de nada lhe interessará o desempenho de uma máquina desktop. Outro fator que foge ao desempenho

é o custo. Se o orçamento de determinado projeto ou usuário for limitado, a arquitetura mais cara e com melhor desempenho não será a mais recomendada, pois o fator determinante da decisão é o valor monetário.

No que concernem as questões de desempenho, não basta verificar unicamente a presença dos componentes individualmente mais potentes ou mais rápidos. A mais alta frequência de *clock*, sozinha, não garante que uma maior quantidade de instruções seja realizada em menor tempo, por exemplo, pois o conjunto de instruções pode não necessariamente estar adequado à sua velocidade, gerando mais calor que eficiência; acarretando a sobrecarga dos componentes próximos que terão manutenção acelerada, maior gasto energético e, por vezes, impossibilitando o *turbo boost* que boa parte das arquiteturas modernas dispõem, em que, por meio do *overclocking*, se permite velocidades maiores em pequenos intervalos de tempo, completando instruções em menos tempo.

Além da análise de velocidade do *clock*, a implementação do conjunto de instruções no *data path*[8] também é crucial para a rapidez na conclusão das tarefas, pois influi diretamente nas possibilidades de como a *pipeline* pode ser explorada ao máximo, mantendo-a cheia e evitando ao máximo a ocorrência de bolhas.

O *benchmark* exposto pode parecer consideravelmente conclusivo à primeira vista, mas é digno de menção que existem variantes cruciais nessas medições que influenciam diretamente no resultado obtido, como os programas que foram executados nos testes - em uma arquitetura suas instruções podem ter um número maior ou menor, o CPI[9] pode ser alto ou baixo e vice-versa; a natureza do teste pode favorecer uma em detrimento da outra - algumas arquiteturas podem ser projetadas visando a utilizar mais memória que registradores, beneficiando a presença de instruções maiores (mais complexas) no código do programa, por exemplo; além de questões externas que são perceptíveis nas comparações de desempenho, como a idade, o desgaste dos componentes de hardware, o quão cheio ou vazio o disco rígido estava, o sistema operacional atuando sobre o hardware da máquina, entre outros fatores.

Assim, mesmo que o *benchmark* seja bastante detalhado, possuindo dados tratados pela empresa e outros mais 'crus', recebidos diretamente por usuários entusiastas; e compondo uma média diante de um espaço amostral numericamente expressivo, não é possível considerá-lo como um método definitivo, preciso e confiável para compração de arquiteturas de computadores. Dessa maneira, compará-las mostra-se realmente uma tarefa complexa e apenas por meio de testes embasados em conhecimento em nível detalhado e com dados acurados seria possível deduzir qual arquitetura se comportará melhor que outra a depender da natureza da utilização desejada.

[8]não incluída nesse documento para a arquitetura i3-2310M dada a carência de informações provenientes de fontes confiáveis.

[9]Ciclos Por Instrução - em média, de um determinado programa

Referências

Gwennap, L. (2010). Sandy bridge spans generations intel focuses on graphics, multimedia in new processor design. *MPR Microprocessor Report*, páginas 1–8.

Intel® (2012). Intel chips. https://www.intel.com.br/content/www/br/pt/history/history-intel-chips-timeline-poster.html. Acesso: 22 Fevereiro 2021.

Intel® (2016). *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Intel Corporation.

Intel® (2019a). Arquivo de produtos intel. https://ark.intel.com/content/www/br/pt/ark/products/52220/intel-core-i3-2310m-processor-3m-cache-2-10-ghz.html. Acesso: 13 Novembro 2019.

Intel® (2019b). *Intel® 64 and IA-32 Architectures Software Developer's Manual*. Intel Corporation.

PassMark® (2019a). Passmark company background. https://www.passmark.com/about/index.php. Acesso: 20 Novembro 2019.

PassMark® (2019b). Passmark cpu performance comparison. https://www.cpubenchmark.net/compare/Intel-i3-2310M-vs-AMD-FX-4100-Quad-Core/756vs255. Acesso: 20 Novembro 2019.

PassMark® (2021). Passmark cpu test information. https://www.cpubenchmark.net/cpu_test_info.html. Acesso: 22 Fevereiro 2021.

Silveira, D. T. e Gerhardt, T. E. (2009). *Métodos de Pesquisa*. Editora da UFRGS.

Tecmundo® (2018). Feliz 50 anos: a história da intel[vídeo]. https://www.tecmundo.com.br/mercado/132281-feliz-50-anos-historia-intel-video.htm. Acesso: 13 Janeiro 2021.