UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Discente: Levy Gabriel da Silva Galvão
Data: 02/07/2020
Disciplina: ELE0621 - ARQUITETURA E PROGRAMAÇÃO DE MICROCOMPUTADORES
Docente: Prof. Dr. José Alberto Nicolau de Oliveira

**Referências**


- Notas de aula.
- CHUANG, Yung-Yu. Lecture notes: Computer Organization and Assembly Languages. 2012.
- DANDAMUDI, Sivarama P. Introduction to Assembly Language Programming For Pentium and RISC Processors. Springer. 2005
- Intel. IA-32 Intel ® Architecture Software Developer's Manual. 2005.
- Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual. 2016
- IRVINE Kip R. Assembly Language for x86 Processors. Prentice Hall. 2015
- PATTERSON, David A. HENNESSY, John L. Computer Organization and Design: The Hardware/Software Interface. Elsevier Inc. 2018.
- STALLINGS. William. Arquitetura e organização de computadores. Pearson. 2018.


**Tópicos**

Processadores híbridos incorporam internamente características RISC com um pequeno conjunto de instruções, mas mantém uma camada de acesso CISC com um grande conjunto de instruções ao usuário. Como se traduz essa interface CISC ao usuário? É a nível de Assembly? (Notas de aula, Introdução aos microprocessadores)

Considerando as diferenças entre a arquitetura de Harvard e de Von Neumann, no caso de uma CPU com pipeline, torna-se mais eficiente utilizar a arquitetura de Harvard, pois esta possui barramentos separados para dados e instruções, uma vez que na arquitetura de Von Neumann só existe um barramento para os dois casos.

Enquanto microprocessadores são sistemas computacionais de propósito geral, capazes de solucionar diversos problemas; microcontroladores como sistemas embarcados que atendem eficientemente aplicações com propósitos específicos; DSPs como uma vertente de microcontroladores especializados em processamento de digital de sinais; o que permite dizer que um FPGA atua como um sistema embarcado de aplicação dedicada? Existe alguma semelhança com aplicação específica? (Notas de aula, Introdução aos microprocessadores)

Como a inclusão do **cache** pós 68020 permitiu executar instruções de acesso a memória com zero estados de espera. (Notas de aula, Microprocessadores - contexto histórico)

O 80486DX incorporou uma unidade de ponto flutuante e uma arquitetura escalar (canalização única) otimizada e uma unidade **caché,** porém qual a diferença entre esta unidade e a cache? (Notas de aula, Microprocessadores - contexto histórico)

As versões de 50 MHz em 5V do 80486DX possuíam problema de superaquecimento, para solucionar o problema a Intel lançou os 486DX2 de 50 MHz e 66 MHz e 3,3V e 486DX4 de 75 MHz, 83 MHz e 100 MHz em 3,3V. Estes multiplicavam segundo um fator 2 ou 3 o clock de processamento enquanto mantinham o clock do barramento no limite de 33 MHz (velocidade máxima dos barramentos ISA, MCA, EISA, VL-BUS ou PCI). A nível de hardware, como é feita a multiplicação de clock?

Altas frequências implicam em uma largura de banda e latência de chips e se tornam um fator limitante, assim a que frequência recomenda-se proteger o circuito gerador de clock? Por exemplo, no caso de um processador Intel Celeron G530 que trabalha a 2.4 GHz com mesma frequência do padrão IEEE 802.11 de Wi-Fi, haveria possibilidade de interferência? (Notas de aula, Microprocessadores - contexto histórico)

Um dos critérios para escolha de um microprocessador é a velocidade (bem como funcionalidade, ISA, capacidade aritmética, lógica e de endereçamento de memória, consumo de energia, custo financeiro, tamanho, tipos de periféricos, ferramentas de software e suporte técnico, disponibilidade de mercado e maturidade do processador). Mas por que não pode ser levada em conta apenas a frequência do clock? Quais outros fatores mais importantes podem afetar a velocidade? Tais como pipeline. (Notas de aula, Microprocessadores - contexto histórico)

O fator principal para a mudança de uma geração para outra seria em termos de organização ao invés de arquitetura, na maioria dos casos?

Qual a razão de usar vários níveis de cache (L1, L2, L3)? Uma vez que os níveis mais internos são mais rápidos. Os níveis de cache só fazem sentido em um processador multicore? Níveis mais externos compartilhados por vários cores servem para os cores utilizarem instruções de cores anteriores? (STALLINGS, 2018, p.5)

Registrador de buffer de instrução (IBR): empregado para manter temporariamente a instrução da direita, da palavra da memória. Esse registrador empregado no IAS (finalizado em 1952) traduz uma versão primitiva do que hoje se conhece como uma memória cache (interpretação própria). (STALLINGS, 2018, p.5)

Evolução da arquitetura Intel x86: (STALLINGS, 2018, p.23-24)

- 8080: máquina de 8 bits e caminho de dados de 8 bits para memória;
- 8086: máquina de 16 bits e caminho de dados mais largo, registradores maiores e fila pré-busca de instruções (primeiro aparecimento da arquitetura x86);
- 80286: endereçamento de uma memória de 16 MB;
- 80386: primeira máquina 32 bits da Intel e aceita **multitarefa**;
- 80486: uso de tecnologia de cache mais poderosa, pipeline sofisticado de instrução, coprocessador matemático, tirando operações matemáticas mais complexas da CPU principal;
- Pentium: uso de técnicas **superescalares**;
- Pentium Pro: ainda com técnicas superescalares e com o uso agressivo de **renomeação de registrador**, **previsão de desvio**, **análise de fluxo de dados** e **execução especulativa**;
- Pentium II: incorporou a tecnologia **Intel MMX**, para processar dados de vídeo, áudio e gráfico eficientemente;
- Pentium III: instruções de ponto flutuante adicionais com a extensão de conjunto de instrução o **Streaming SIMD Extensions** e 70 novas instruções projetadas para aumentar o

desempenho da repetição de operações em objetos de dados múltiplos;

- Pentium 4: ponto flutuante adicional e melhorias para multimídia.
- Core: primeiro microprocessador Intel x86 dual core;
- Core 2: arquitetura de 64 bits; o Core 2 Quad oferece quatro processadores em um único chip; o conjunto de instruções **Advanced Vector Extensions** contribui para um processamento eficiente de vetores de dados de 256 bits e 512 bits.

Dentre as técnicas mais comuns para o aumento de velocidade do processador, estão: realização de pipeline, execução superescalar, predição do desvio a partir da análise do código da instrução, análise de fluxo de dados para detectar quais instruções necessitam dos resultados de outras e por meio das duas últimas técnicas, processadores são capazes de, a partir da execução especulativa, executam especulativamente instruções que não surgiram na execução do código e armazena os resultados temporariamente. (STALLINGS, 2018, p.39)

Processadores com técnicas **superescalares** possuem as mesmas características que uma pipeline, mas são capazes de iniciar várias instruções simultaneamente e executá-las ao mesmo tempo e independentemente. Porém limitações como: conflito de recursos entre instruções, desvios de código e dependência de dados, resultam em maior severidade que uma pipeline normal. Outra alternativa é o **superpipeline** que divide os estágios de uma pipeline em sub-estágios, cada um executado em meio ciclo de clock, aumentando o número de instruções suportadas. Dessa forma um fator limitante é o conflito de dados e os desvios de código, que são tão prejudiciais quanto o tamanho da pipeline. (STALLINGS, 2018, p.496)

A velocidade com que os dados são transferidos da memória para o processador é menor que a crescente velocidade do processador. Soluções envolvem alteração no chip da DRAM com maior largura do número de bits, ou uso de cache para evitar acessos à memória, ou até barramentos rápidos que podem ser capazes de armazenar e estruturar o fluxo de dados. (STALLINGS, 2018, p.39-40)

Deve-se pensar no equilíbrio de demandas de fluxo e processamento de componentes por meio da taxa de desempenho diferenciada para diversas áreas da tecnologia e novas aplicações e dispositivos. (STALLINGS, 2018, p.40)

O aumento da velocidade do processador deve-se a: aumento da velocidade de seu hardware; maior tamanho e velocidade dos caches entre o processador e a memória; alterar a arquitetura e organização do processador maior velocidade na execução das instruções. (STALLINGS, 2018, p.40-41)

Os fatores limitadores são: maior potência causada pela densidade da lógica e velocidade de clock, implicando em necessidade de dissipar calor; atrasos devido à natureza resistiva e capacitiva devido as conexões; velocidade de acesso à memória (latência) e taxa de transferência de memória. (STALLINGS, 2018, p.41)

O uso de chips multicores levou ao uso de vários níveis de cache, com o primeiro para um só processador e o segundo e terceiro são compartilhados entre processadores. Às vezes se torna comum o segundo nível de cache ser privado a um processador. (STALLINGS, 2018, p.41)

(PATTERSON & HENNESSY, 2018) sugerem oito grandes ideias para arquitetura de computadores: projeto visualizando a lei de Moore; usar níveis de abstração para simplificar o projeto; tornar rápido o caso mais frequente; desempenho via paralelismo, pipeline e predição;

confiabilidade por meio de redundância; e hierarquia de memória, mantendo as menores, mais rápidas e mais caras no topo e as maiores, lentas e baratas na base (**o que significa manter no topo ou na base? seria em relação à proximidade ao processador?**). (PATTERSON & HENNESSY, 2018, p.11-12)

Para determinar o desempenho da CPU, pode-se recorrer à fórmula clássica para encontrar o tempo de CPU, calculada pelo produto entre o número de instruções executadas pelo programa e a quantidade de ciclos por instrução, divididos pela taxa de clock. (PATTERSON & HENNESSY, 2018, p.36)

Modos de operação e capacidade de endereçamento: (IRVINE, 2015, p.37-38)

- **Modo protegido** - nativo; todas instruções e recursos disponíveis; programas possuem segmentos separados de memória; processador previne de programas referenciar memórias fora do segmento estipulado;
- **Modo virtual-8086** - executa múltiplas sessões virtual-8086 ao mesmo tempo;
- **Modo de endereçamento real** - implementa um ambiente de um processador Intel primitivo com mais recursos, como mudar para outros modos; útil se um programa requer acesso direto à memória do sistema and dispositivos de *hardware*;
- **Modo de gerenciamento do sistema** (SMM) - provê um OS com mecanismos para implantar funções como gerenciamento de energia e segurança do sistema;

Dentre os modos de operação da arquitetura x86 (modo protegido, de endereço real, de gerenciamento de sistema e virtual-8086), o modo virtual-8086 é um híbrido do modo protegido, no qual cada programa possui seu próprio computador 8086 para gerenciamento. Em que sentido se tratam esses programas, eles poderiam ser considerados como navegadores, visualizadores de mídia, etc? Como são tratados fisicamente os processadores 8086 virtuais? (CHUANG, 2012, p.10)

O modo de gerenciamento do sistema seria aquele em que a BIOS é executado? (IRVINE, 2015, p.38)

Como mudar de um modo de operação para outro? Só pode ser feita durante o *boot*, ou tem alguma possibilidade de ser feita após o SO ser inicializado? (IRVINE, 2015, p.38)

Tabela 1 - Registradores básicos para a execução de programas na arquitetura x86. (DANDAMUDI, 2005), (Intel, 2005), (Intel, 2016) e (CHUANG, 2012).

| Grupo | Tipo | Função | 32-bits | 16-bits | 8-bits | |
|---|---|---|---|---|---|---|
| | Dados | Acumulador para operandos e resultados | EAX | AX | AH | AL |
| | | Ponteiro para dado no segmento DS | EBX | BX | BH | BL |
| | | Contador para operações com *loops* e *strings* | ECX | CX | CH | CL |
| | | Ponteiro de E/S | EDX | DX | DH | DL |

| | | | | | |
|---|---|---|---|---|---|
| Propósito geral | Indexa-dor | Ponteiro para dados no segmento apontado pelo registrador DS; ponteiro fonte para operações com *string* | ESI | SI | -- |
| | | Ponteiro para dados (ou destino) no segmento apontado pelo registrador ES; ponteiro destino para operações com *string* | EDI | DI | -- |
| | Apon-tador | Ponteiro de pilha (no segmento SS) | ESP | SP | -- |
| | | Ponteiro para dado na pilha (no segmento SS) | EBP | BP | -- |
| Controle | -- | Ponteiro para o endereço da instrução | EIP | IP | -- |
| | -- | *Flags* de estado, controle e sistema | EFLAGS | FLAGS | -- |
| Segmento | -- | Aponta para o local na memória principal onde as instruções de programa estão armazenadas | -- | CS | -- |
| | -- | Aponta para o local na memória principal onde os dados estão armazenados | -- | DS | |
| | -- | Aponta para o segmento de pilha de programa | -- | SS | |
| | -- | Sobressalentes, capazes de serem utilizados como os outros três, i.e: uso de dois segmentos para um dado de programa que supera um segmento | -- | ES | -- |
| | -- | | -- | FS | -- |
| | -- | | -- | GS | -- |

Tabela 2 - *Flags* da arquitetura x86. (DANDAMUDI, 2005).

| Tipo de *flag* | Descrição | Sigla | Descrição | Sigla |
|---|---|---|---|---|
| Estado | Carry | CF | Zero | ZF |
| | Parity | PF | Sign | SF |
| | Auxiliary carry | AF | Overflow | OF |
| Sistema | Trap | TF | Virtual 8086 mode | VM |
| | Interrupt | IF | Alignment check | AC |
| | I/O privilege level | IOPL | Virtual interrupt | VIF |
| | Nested task | NT | Virtual interrupt pending | VIP |

| | | | | |
|---|---|---|---|---|
| | Resume | RF | ID | ID |
| Controle | Direction | DF | -- | -- |

No modo protegido, o endereço lógico é aquela gerado pela CPU para referenciar um espaço de memória, porém ele não existe fisicamente, sendo necessário ser traduzido para um endereço linear pela tradução de segmento (segment translation) e depois para um endereço físico por meio da tradução de página (page translation). De forma que este último identifica um local físico de memória de dados. Por que que a necessidade desta última tradução? (DANDAMUDI, 2005, p.54)

Qual a necessidade do *paging* na arquitetura de memória do modo protegido? Já que o endereço linear gerado durante a segmentação pode ser tratado como o físico, caso nenhum mecanismo de *paging* seja usado. (DANDAMUDI, 2005, p.54)

A parte visível dos **registradores de segmento** são os 16-bits do seletor de segmento (*segment selector*), e fornece: um **índice** (13-bits) para selecionar um descritor de segmento (*segment descriptor*) da tabela de descritor local ou da tabela de descritor global; bit **indicador de tabela** (TI) que aponta qual tabela será usada; o **nível de privilégio do solicitante** (RPL) delimita as permissões de acesso a dados protegido. (DANDAMUDI, 2005, p.54-55)

Já o **descritor de segmento** provê os atributos de um segmento, tais como: **endereço base** (32-bits) para o início do segmento na memória de 4GB; bit **granularidade** (G) define se os tamanhos dos segmentos serão de 1 byte (0) ou 4 Kbyte (1); **limite de segmento** (20-bits) que pode ser de 1 byte até 1 Mb (0) ou de 4 Kb até 4 Gb (1), a depender da granularidade; **bit D/B**, em segmento de código (D) define o tamanho padrão dos operandos e *offsets*, 16-bits (0) ou 32-bits (0) e em segmento de dados (B) ele controla o tamanho de pilha e ponteiro de pilha, pode usar o registrador SP (0) ou ESP (1) para operações de pilha; **bit S** identifica se o segmento é de sistema (0) ou aplicação (código ou dados) (1); o campo **nível de privilégio do descritor** (DPL) define o nível de privilégio do segmento; o campo **tipo** identifica o tipo de segmentos a depender do bit S, e em caso de segmento de aplicação, depende se é de código **tipo** identifica como *execute-only*, *execute/read-only*, etc.) ou dados (**tipo** identifica como *read-only*, *read-write*, etc.); o **bit P** identifica a presença do segmento, se 0, o processador indica a não presença do segmento quando um seletor para o descritor for carregado em um registrador de segmento. (DANDAMUDI, 2005, p.55-57)

Uma **tabela de descritor de segmento** (de 8 bytes a 64 Kb) é um vetor de descritores (código e dados) de segmento, e existem três tipos: uma tabela de descritor global (GDT) com descritores para todas as tarefas no sistema, várias tabelas de descritor local (LDT) que contém descritores (para código, dado, pilha, etc.) para um dado programa e a tabela do descritor de interrupção (IDT) cujas entradas são consultadas em busca do procedimento para lidar com interrupções do sistema. Cada uma possui um registrador (LDTR, GDTR) que guarda um endereço base linear de 32-bits e um tamanho de tabela de 16-bits. (DANDAMUDI, 2005, p.57-58) (Intel, 2005, p.6-14)

No modo de endereçamento real, para cada endereço lógico de memória só existe um endereço físico, mas mais de um endereço lógico pode se referir ao mesmo endereço físico. Pode-se fazer uma analogia com as condições existência de uma função $f(x)$, que para para $f(x)$ pertencente ao domínio da função, só existe uma $f(x)$ correspondente no contradomínio e que múltiplos $x$ podem apontar para um mesmo $f(x)$. A figura 1 ilustra. Isso se deve ao fato de registradores de 16-bits terem que representar endereços de 20-bits (DANDAMUDI, 2005, p.61)

No que diz respeito à alocação física em modo real, a alocação de um segmento pode se apresentar totalmente ou parcialmente dissociada ou totalmente sobreposto a outra. Neste contexto qual a necessidade de haver segmentos parcialmente ou totalmente sobrepostos? (Notas de aula - Arquitetura x86, p.24)

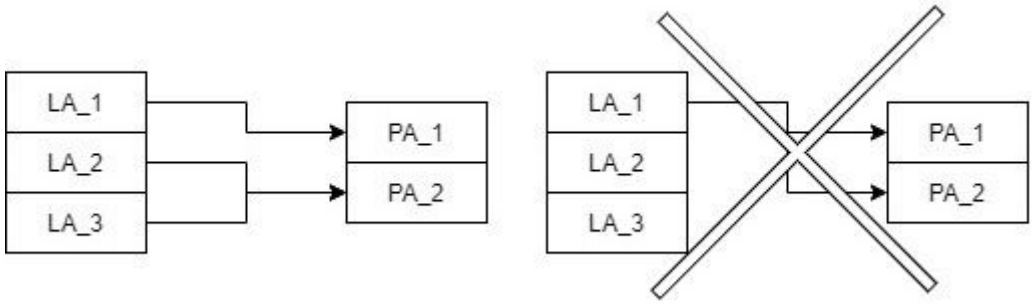



Figura 1 - *Logical address* (LA) e modo de se referenciar a *physical address* (PA). (Fonte própria)

No endereçamento de memória, se rotinas são definidas antes do código principal do programa, o CS:IP irá apontar para o início do programa em endereços de memória diferentes dos iniciais, sendo estes reservados a essas funções. O acesso dessas funções se dá pela pilha?

**Resumo das perguntas**

Como se traduz essa interface CISC ao usuário? É a nível de Assembly? R.: A interface CISC híbrida com RISC se traduz, principalmente, a nível de Assembly, pois permite uma grande número de instruções que, internamente, são transformadas em instruções mais simples, evitando um grande número de instruções que de fato são implementadas e utilizadas raramente.

O que permite dizer que um FPGA atua como um sistema embarcado de aplicação dedicada? Existe alguma semelhança com aplicação específica? R.: Dedicado se refere ao fato do sistema ser encapsulado ou dedicado ao dispositivo ou sistema que ele controla.

O fator principal para a mudança de uma geração para outra seria em termos de organização ao invés de arquitetura, na maioria dos casos? R.: É um fator que depende do fabricante e, em certas gerações pode-se alterar a arquitetura e/ou a tecnologia, não se limitando a uma ou outra.

Diferença entre cache e caché? R.: Cache refere-se a uma memória mais externa que caché.

A nível de hardware, como é feita a multiplicação de clock? R.: Circuito do tipo Phase-Locked Loop (PLL) são capazes de multiplicar o clock externo para que o clock interno da CPU seja mais rápido.

A que frequência recomenda-se proteger o circuito gerador de clock? R.: Ao passo que a velocidade do processador aumenta, podendo se traduzir pelo aumento da frequência do clock, há a necessidade de proteger o circuito contra interferência eletromagnética, principalmente ao considerar que a frequência no nível de gigahertz ficou cada vez mais comum em processadores e possíveis fontes de interferência.

Qual a razão de usar vários níveis de cache (L1, L2, L3)? Uma vez que os níveis mais internos são

mais rápidos. Os níveis de cache só fazem sentido em um processador multicore? Níveis mais externos compartilhados por vários cores servem para os cores utilizarem instruções de cores anteriores? R.: Levando em conta a hierarquia de memória, vários níveis de cache pode permitir trabalhar com diferentes tamanhos de memória e velocidades, levando em conta que memórias mais externas possuem uma menor frequência de acesso e as internas com maior frequência.

Em uma hierarquia de memória, o que significa manter no topo ou na base? Seria em relação à proximidade ao processador? R.: Memórias mais rápidas devem estar no topo e mais próximas ao processador, permitindo uma maior velocidade no acesso do conteúdo da memória, permitindo agilizar algumas tarefas,, enquanto que memórias mais lentas e maiores ficam reservadas a um armazenamento cuja a frequência de acesso dos dados é menor.

Durante o modo protegido existe uma tradução do endereço lógico para linear e de linear para físico. Porém quando a última tradução é inexistente, o endereço linear é considerado o físico, mas por que a necessidade desta tradução? R.: A paginação permite dividir a memória em tamanhos iguais, facilitando o gerenciamento, enquanto que a segmentação divide a memória em pedaços de tamanhos arbitrários.

Em que sentido se tratam esses programas, eles poderiam ser considerados como navegadores, visualizadores de mídia, etc? Como são tratados fisicamente os processadores 8086 virtuais? R.: Sim. O processador considera uma quantidade de bits maior virtualmente para gerenciamento dos processadores virtuais.

O modo de gerenciamento do sistema seria aquele em que a BIOS é executado? R.: Sim.

Como mudar de um modo de operação para outro? Só pode ser feita durante o *boot*, ou tem alguma possibilidade de ser feita após o SO ser inicializado? R.: O gerenciamento de modos de operação é automático, pois os microprocessadores trabalham normalmente em modo protegido e no caso de uma emulação ele automaticamente considera o modo real.

Qual a necessidade do *paging* na arquitetura de memória do modo protegido? R.: O *paging* permite acessar blocos de memória de 4 KB.

Qual a necessidade de haver segmentos parcialmente ou totalmente sobrepostos? R.: Na manipulação de vetores, eles só podem podem ser movidos para diferentes endereços lógicos, de forma que segmentos diferentes sobrepostos podem apontar para diferentes endereços físicos, permitindo mover o vetor entre endereços físicos, mantendo o mesmo endereço lógico.

No endereçamento de memória, se rotinas são definidas antes do código principal do programa, o CS:IP irá apontar para o início do programa em endereços de memória diferentes dos iniciais, sendo estes reservados a essas funções. O acesso dessas funções se dá pela pilha? R.: A pilha armazena automaticamente o contexto do programa para que após executar uma rotina retornar com segurança para o ponto de código que foi interrompido pela função. Dessa forma, apenas próprias instruções do programa pode fazer desvio para a função e não endereços na pilha.

O tipo TBYTE suporta inteiros de 80-bit. Para quê são usados? De que forma esses tipos são armazenados em hardware? (CHUANG, 2012, p.15) R.:Rotinas em ponto flutuante que ocupam 10 bytes, ou seja, 10 posições de memória (com o menos significativo em 1).

Qual a diferença entre os atributos *far* e *near* de um procedimento? (Notas de aula, Linguagem

"assembly" para o IBM-PC) R.: O atributo *nea*r trabalha com um procedimento no mesmo segmento de memória, enquanto o *far* diz respeito a outros segmentos, de forma que procedimentos de bibliotecas devem ser criados com o atributo *far*.

Loops muito grandes geram erros. R.: Isso pois um endereço de 8-bit com sinal é usado para indicar o *jump,* podendo se deslocar em 127 posição para baixo ou para cima, sendo esta a quantidade limitante de posições de desvio condicional. (Notas de aula, Linguagem "assembly" para o IBM-PC)

Como trabalhar inserindo via teclados número com mais de um dígito e como representá-los em console? R.: Recebe os dígitos mais significativos e multiplica por 10 para logo após somar com os novos dígitos inseridos.

**Exercícios propostos**

**1**. uP x86 com 4 níveis de *pipeline*, *clock* de 2.25 Ghz e qualquer ciclo de barramento ocorre a um período de *clock*.

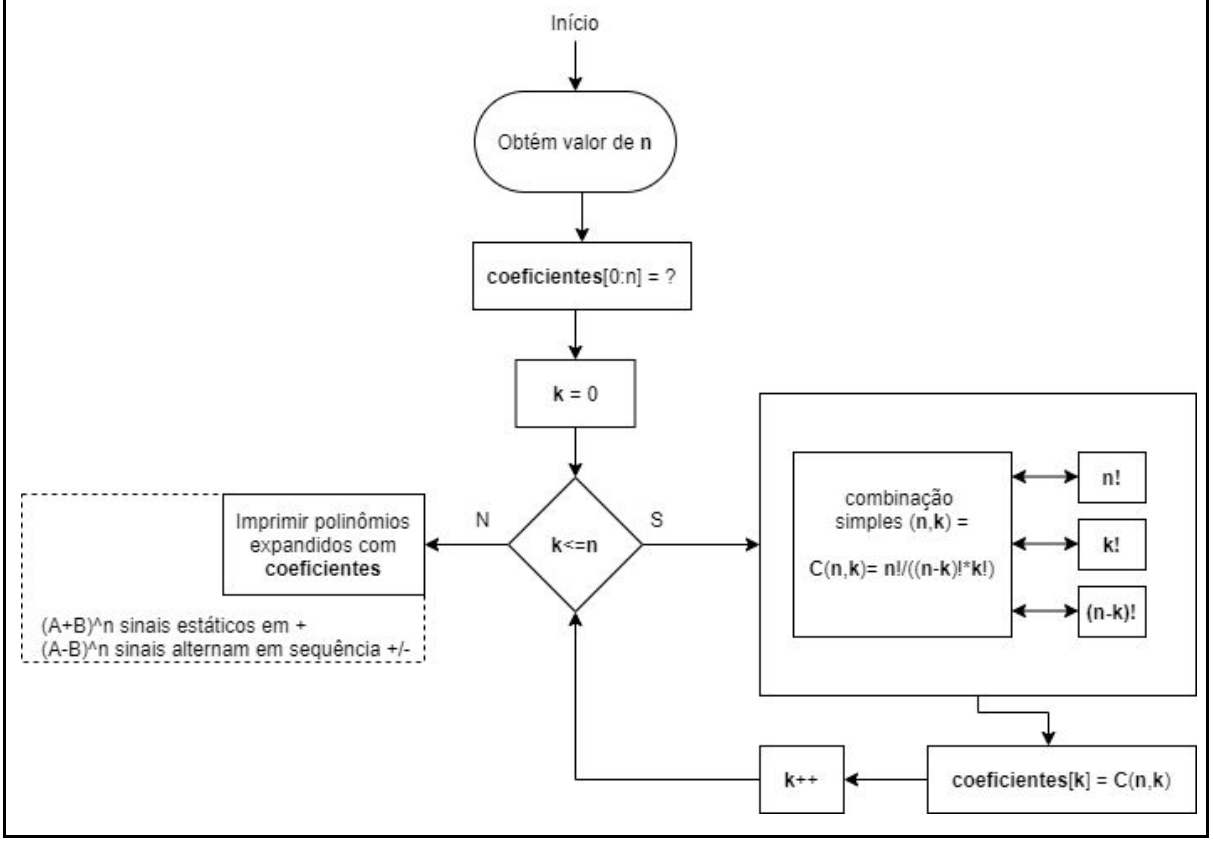
I1 = UB, UI, UA, UB, UE, UA, UB
I2 = UB, UI, UE, UA, UB
I3 = UB, UI, UA, UE

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| UB | I1 | I2 | I3 | I1 | X | I2 | I1 | X | I1 | I2 | X | I1 | X | I2 | I1 |
| UI | X | I1 | I2 | I3 | I3 | I3 | X | X | X | I1 | I2 | X | X | X | X |
| UE | X | X | X | I2 | I1 | X | X | I3 | X | X | X | I2 | I1 | X | X |
| UA | X | X | I1 | X | I2 | I1 | I3 | X | X | X | I1 | X | I2 | I1 | X |
| Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Na estrutura *pipeline* proposta, as instruções I1 e I2 são executadas em 7 ciclos ou 3.11 ns. Se a instrução I3 fizer com que as instruções I1 e I2 sejam repetidas em um *loop* e considerando que o *loop* fosse executado no mínimo uma vez, ele teria uma duração de 7 ciclos, de forma que o sistema sairá do *loop* em 3.11 ns.

A taxa média de ciclos por instrução (CPI) para a estrutura *pipeline* é de 3 ciclos por instrução (15 ciclos divididos por 5 instruções). Enquanto que o CPI para uma estrutura sequencial que executa essas mesma instruções é de 5.6 ciclos por instrução (7+5+4+7+5 por 5 instruções). O ganho da estrutura *pipelined* sobre a sequencial é de 1.8667.

**Desafio.**


Início
Obtém valor de n
coeficientes[0:n] = ?
k = 0
k<=n
N
S
Imprimir polinômios expandidos com coeficientes
(A+B)^n sinais estáticos em +
(A-B)^n sinais alternam em sequência +/-
combinação simples (n,k) =
C(n,k)= n!/((n-k)!*k!)
n!
k!
(n-k)!
coeficientes[k] = C(n,k)
k++