Dênis Araújo da Silva

Marcos Aurélio Freitas de Almeida Costa

Nicolas Luiz Ribeiro Veiga

Desenvolvimento de um Microprocessador 8086 RISC em FPGA

Itajubá - MG

06 de novembro de 2013

Dênis Araújo da Silva

Marcos Aurélio Freitas de Almeida Costa

Nicolas Luiz Ribeiro Veiga

Desenvolvimento de um Microprocessador 8086 RISC em FPGA

Documento apresentando as normas gerais para o desenvolvimento e a redação do trabalho de diploma do curso de Engenharia da Computação da Universidade Federal de Itajubá.

Universidade Federal de Itajubá - UNIFEI
Instituto de Engenharia de Sistemas e Tecnologias da Informação
Engenharia da Computação

Itajubá - MG

06 de novembro de 2013

Sumário

Lista de Figuras

Lista de Tabelas

1 Resumo

Este documento descreve o Trabalho Final de Graduação do curso de Engenharia da Computação da Universidade Federal de Itajubá.

O projeto visa desenvolver um microprocessador iAPX86 de arquitetura RISC implementado em linguagem VHDL e simulado em um kit FPGA Altera DE1.

2 Introdução

O microprocessador, ou simplesmente CPU, é uma peça fundamental dos dispositivos eletrônicos atuais. Esta peça está presente em computadores pessoais, tablets, smartphones e eletrodomésticos. É responsável pela execução de operações aritméticas e lógicas requisitadas pelos programas.

O projeto de um microprocessador envolve circuitos grandes e complexos, é neste ponto que entra a lógica programável. A utilização desta permite escrever um código que implemente a funcionalidade de um circuito eletrônico. VHDL é uma das linguagens que permite a escrita deste código, sendo independente de tecnologia e fabricante.

Um Dispositivo Lógico Programável, ou PLD, é um hardware fixo que pode ser configurado para atender a uma determinada funcionalidade. A tecnologia FPGA é um exemplo de PLD, e possui inúmeros chips que podem ser programados para executar diversas funçoes, desde controladores de vídeo até processadores simples.

Este trabalho tem como objetivo desenvolver os blocos de um microprocessador arquitetura x86 em VHDL e testar sua funcionalidade em uma FPGA.

3 Objetivo

O objetivo deste trabalho é adaptar o microprocessador 8086 para um dispositivo de arquitetura RISC Load/Store. O dispositivo será implementado em linguagem VHDL e testado em uma FPGA.

4 Fundamentação Teórica

4.1 Microprocessadores

4.1.1 Definição

Segundo (**??**), um microprocessador é um circuito eletrônico muito complexo composto de milhares de transistores microscópicos num único circuito integrado contendo até cerca de 40 terminais. Os milhares de transistores que compõem o microprocessador são arranjados para formar muitos circuitos diferentes dentro do chip. Entre estes circuitos pode-se destacar registradores, decodificadores, contadores, etc.

O coração de um microcomputador é sua unidade de processamento (MPU). A MPU de um microcomputador é implementado com um dispositivo VLSI (*Very Large Scale Integration*) conhecido como microprocessador, ou somente processador, sendo mais direto. Um microprocessador é uma unidade de processamento de propósito geral construído em um único circuito integrado (CI), (**??**).

Como visto acima sabemos que o microprocessador é o coração de um sistema microprocessado, na figura 2, defini-se as quatro partes básicas de um sistema microprocessado, que incluem um microprocessador, memória e entrada/saída que são interligados por um sistema de *buses*, que será explicado mais a frente. Um *bus* é um conjunto de fios que transmitem informação entre dois ou mais dispositivos(**??**).

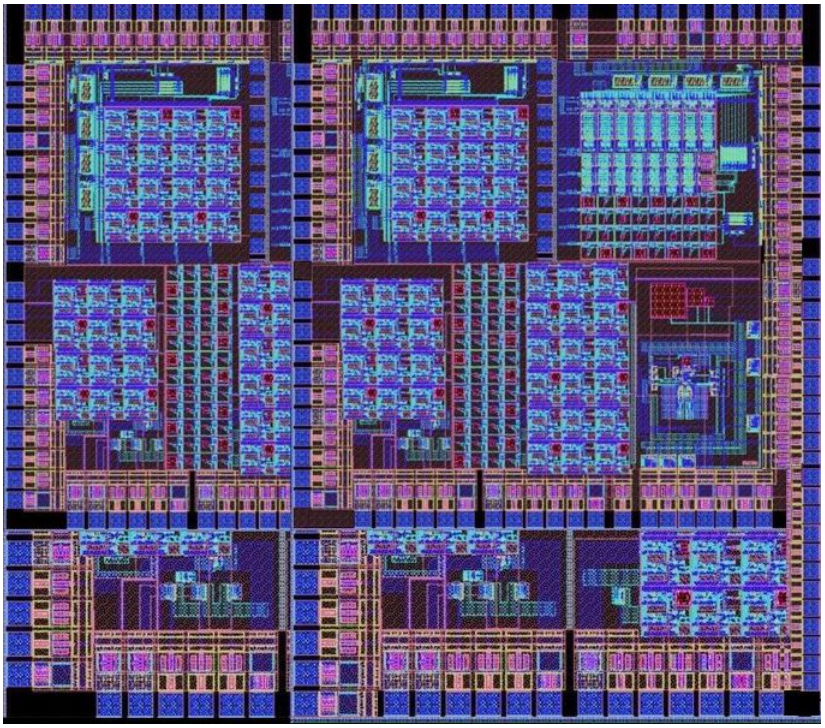

Figura 1: Microcircuito produzido pelo processo fotográfico de multicamadas. Microprocessador com frequência de 4.8GHz, utilizado para o processamento de imagens do Gyroscan 6,2 Tesla. (**??**)

4.1.2 Funcionamento

Os microprocessadores funcionam à partir de um relógio interno, feito de quartz que quando sujeito a uma corrente elétrica, emite pulsos, chamados de ”top”. Tais pulsos fazem com que o microprocessador execute uma ação, ou seja, uma instrução, seja a mesma executada de forma parcial ou total. Estes pulsos também define a potência do microprocessador, sendo esta potência é definida como o número de instruções executadas pelo por segundo e tem como unidade utilizada o MIPS (Milhões de Instruções Por Segundo) (**??**).

Existem dispositivos de entrada e saída que permite a importação de dados para armazenamento ou processamento, a exportação dos resultados e acessos aos dados armazenados. Outros sinais importantes para o funcionamento do micro-

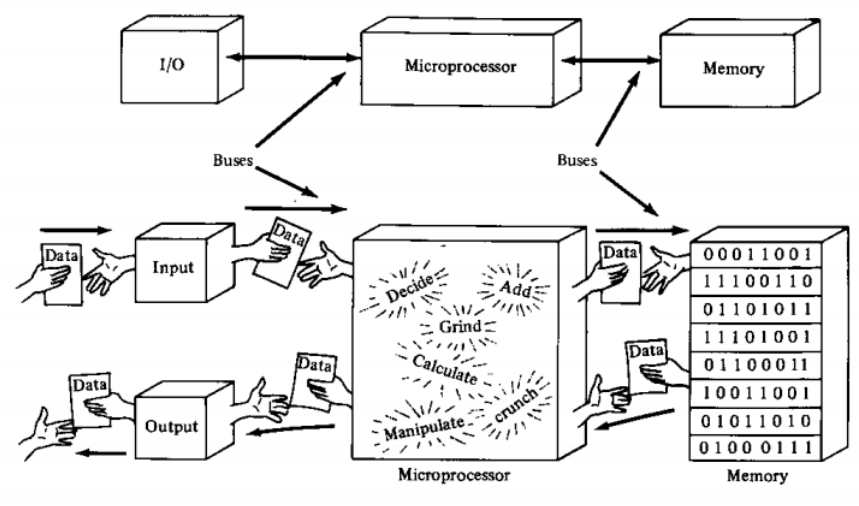

Figura 2: Diagrama de blocos de um sistema microprocessado. (**??**)

processador é o sinal de *reset*, que faz com que a CPU volte a um estado inicial que é definido e conhecido, voltando a este estado o microprocessador pode começar a executar programas, e o sinal de interrupção que faz com que o microprocessador pare sua execução e comece a executar uma rotina pré-definida (**??**)

4.1.3 Programa de Computador

A figura 3 é uma representação visual de um programa de computador, onde ter-se o código não é o suficiente. Para realizar a tarefa especificada pelo programa, o computador (microprocessador) necessita ler as instruções do programa, interpretá-las e executá-las. (**??**).

De acordo a (**??**) a maneira que um computador executa um programa é cíclica e segue a seguinte ordem:

1. Leitura (*Fetch*) de uma instrução
   Onde o computador lê uma instrução e copia a instrução da memória para o seu cérebro (*Microprocessador*).

2. Interpretação (*Decode*) da instrução

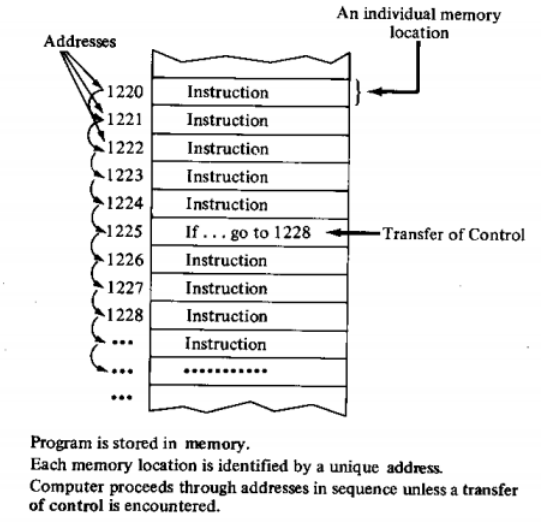

Figura 3: Organização de um programa de computador (**??**)

Cada número que representa uma instrução dentro do programa, possui um certo significado para o computador, em termos de qual ação que deve ser realizada.

3. Execução (*Execute*) da instrução

Para a realização deste ciclo, o microprocessador conta com uma série de circuitos internos com funcionalidades específicas, que serão descritos à seguir.

4.1.4 Registradores

Os registradores, são utilizados para salvar informação binária durante o tempo de execução de um programa. Cada registro possui uma função específica associada a ele (**??**):

O **acumulador**, é um registro primário associado a uma ALU (Unidade Lógica Aritmética) e as operações de entrada/saída. O **registro de instrução**, guarda o código binário da instrução que está sendo executada. O **contador de programa**, contém o endereço de memória da próxima instrução que deve ser tomada.

Todos estes registradores podem ser visualizados na figura 4.

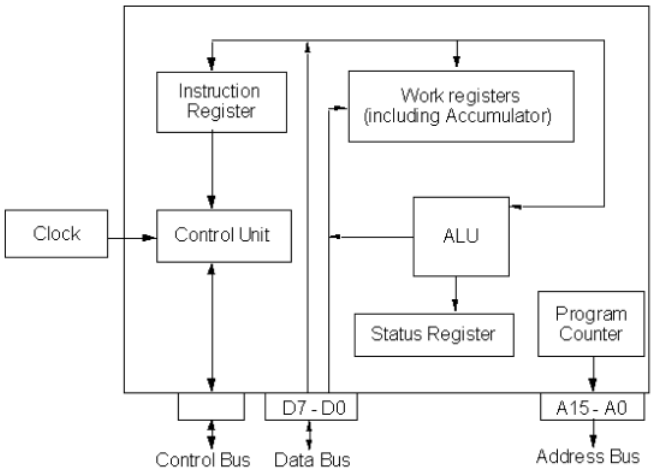

Figura 4: Organização interna de um microprocessador hipotético (**??**)

4.1.5 Unidade Lógica Aritmética

A Unidade Lógica Aritmética, ou *Arithmetic Logic Unit* (ALU) em inglês, é o maior componente da unidade central de processamento de um sistema microprocessado. A unidade realiza todos os processos relacionados a operações aritméticas e lógicas que necessitam ser feitas nas instruções. Em alguns microprocessadores a ALU é divida em unidade aritmética (UA) e unidade lógica (UL). Uma ULA pode ser desenvolvida por engenheiros para calcular qualquer operação. Assim que as operações começam a ficar mais complexas, a ULA fica mais cara, ocupa mais espaço e dissipa mais calor. Por isto engenheiros fazem a ULA poderosa o suficiente, para garantir que a Unidade de Processamento seja também poderosa e rápida, porém não tão complexa, que a torne proibitiva em termos de custo entre outras desvantagens (**??**).

ULAs normalmente realizam as seguintes operações:

- **Operações Lógicas**: Essas incluem AND, OR, NOT, XOR, NOR, NAND,

etc.

- **Operações de Rotacionamento de Bits**: Pertence ao rotacionamento da posição dos bits por um certo número de rotações para direita ou esquerda, que são consideradas operações de multiplicação.

- **Operações Aritméticas**: Refere-se a adição e subtração de bit. Entretanto multiplicação e divisão são as vezes implementadas, essas operações são mais caras de se realizar. A adição pode ser utilizada como substituta para a multiplicação e a subtração para a divisão.

4.1.6 Unidade de Controle

A unidade de controle é composta por um controlador de sequência e um decodificador de instrução. Durante a execução, a unidade de controle, ajusta o conteúdo do contador de programa para ser posicionado nas linhas de endereçamento. Essas linhas indicam o endereço da posição de memória, que contém o código da próxima instrução a ser executada. Em seguida, a unidade de controle insere no registrador de instrução o código de instrução da posição de memória. O decodificador de instrução é habilitado e a unidade de controle ativa as linhas de controle necessárias, buscando dos resultados desejados (**??**).

4.1.6.1 Sinais de Controle

Os sinais de controle são sinais elétricos que orquestram as diversas unidades do processador, que participam na execução de uma instrução. Os sinais de controle são distribuídos devido a um elemento chamado sequenciador. O sinal *Read/Write*, em português Leitura/Escrita, diz para a memória ou outros dispositivos que o processador quer ler ou escrever uma informação (**??**).

4.1.7 Sistema de Barramentos

No diagrama simplificado da figura 5 todos os módulos lógicos se comunicam com a Unidade Central de Processamento. Na prática, muitos modelos de interconexão podem ser usados, geralmente através de barramentos. Lembre-se de que um barramento é um meio de transmissão de informações ou sinais, distinguidos por suas funções. No caso dos sistemas baseados em microprocessador, ao menos três barramentos são fornecidos (**??**):

- **Barramento de Dados**: Transmite dados entre as unidades. Portanto, um microprocessador de 8 bits requer um barramento de dados de 8 linhas para transmitir dados de 8 bits em paralelo. Semelhantemente, um microprocessador de 64 bits necessita de um barramento de dados de 64 linhas para transmitir dados de 64 bits em paralelo. Se o barramento de dados para um microprocessador de 64 bits fosse formado por 8 linhas, seriam necessárias oito transmissões sucessivas, tornando mais lento o sistema. O Barramento de Dados é bi-direcional, isto é, pode transmitir em ambas as direções.

- **Barramento de Endereço**: É usado para selecionar a origem ou destino de sinais transmitidos em um dos outros barramentos ou numa de suas linhas, conduzindo endereços. Uma função típica do Barramento de Endereço é selecionar um registrador em um dos dispositivos do sistema, que é usado como a fonte ou o destino do dado. O Barramento de Endereço do nosso computador padrão, tem 16 linhas e pode endereçar 216 (64 K) dispositivos (1K = 1024, ou 210 , no jargão de computação).

- **Barramento de Controle**: Sincroniza as atividades do sistema, conduzindo o status e a informação de controle de/para o Microprocessador. Para um Barramento de Controle ser formado, ao menos 10 (geralmente são mais) linhas de controle são necessárias.

De acordo a (**??**), os barramentos são implementados como linhas de comunicação reais. Eles podem ser posicionados como parte do circuito no próprio Chip (Barramentos internos) ou podem servir de comunicação externa entre os Chips (Barramentos externos). Os barramentos externos podem ser expandidos para facilitar a conexão de dispositivos especiais. Um projeto eficiente de barramentos é crucial para a velocidade do sistema.

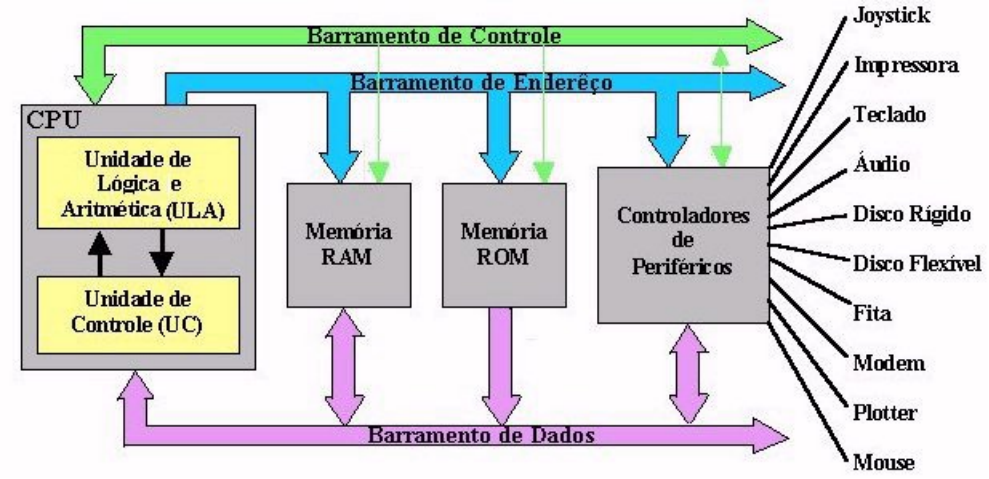

Figura 5: Diagrama simplificado de um Microcomputador (**??**)

4.1.8 Dispositivos de Entrada/Saída

Os dispositivos de entrada/saída (E/S) ou *input/output (I/O)*, são também denominados periféricos, eles permitem a interação do processador com o homem, possibilitando a entrada e/ou saída de dados. Porém, é necessário o módulo mais a direita da figura 5, que são os controladores de periféricos. Tais controladores possuem a tarefa de combinar as velocidades entre os dispositivos, pois a maioria dos periféricos são consideravelmente mais lentos que a unidade de processamento, convertem dados de um formato em outro (**??**). Exemplos de periféricos de entrada: teclado, mouse, scanner, etc. Dispositivos de saída: monitor, impressora, etc.

4.1.9 Arquiteturas

De acordo com (**??**), uma das mais importantes abstrações é a interface entre o hardware e o software de baixo nível. Por causa de sua importância, é dado uma nomenclatura especial: **arquitetura do conjunto de instruções** (ISA), ou simplesmente arquitetura de uma máquina. O conjunto de instruções, inclui qualquer coisa que programadores necessitam para saber como programar em linguagem de máquina corretamente, incluem instruções, dispositivos E/S, entre outros. Tipicamente o sistema operacional irá encapsular os detalhes da realização da E/S, alocação de memória, e outras funcionalidades de baixo nível do sistema, portanto, programadores não precisam se preocupar com estes detalhes. Dois tipos de conjuntos de instruções existentes serão explicados a diante.

4.1.9.1 CISC - Complex Instruction Set Computer

CISC é uma arquitetura de processador, que teve como princípio o uso eficiente de memória e a facilidade de programar. Cada instrução desse processador tem várias operações em seu interior ajudando o programador a implementar programas. A maioria dos projetos de microprocessadores comuns - incluindo o Intel (R) 80x86 e séries Motorola 68K - também seguem a filosofia CISC (**??**).

Os primeiros processadores utilizados para decodificar e executar cada instrução no conjunto de instrução do processador, principalmente para trabalhos simples, com poucos registros, funcionou, mas não para sistemas complexos. Assim, os seus criadores construíram uma lógica simples de controlar os caminhos de dados entre os vários elementos do processador, e usou um conjunto simplificado de instruções de microcódigo para controlar a lógica do caminho de dados.

A microprogramação é uma representação simbólica do controle em forma de instruções, chamadas microinstruções, que são executadas em uma micromáquina simples (**??**), podemos ver na figura 6 o exemplo de um microcódigo.

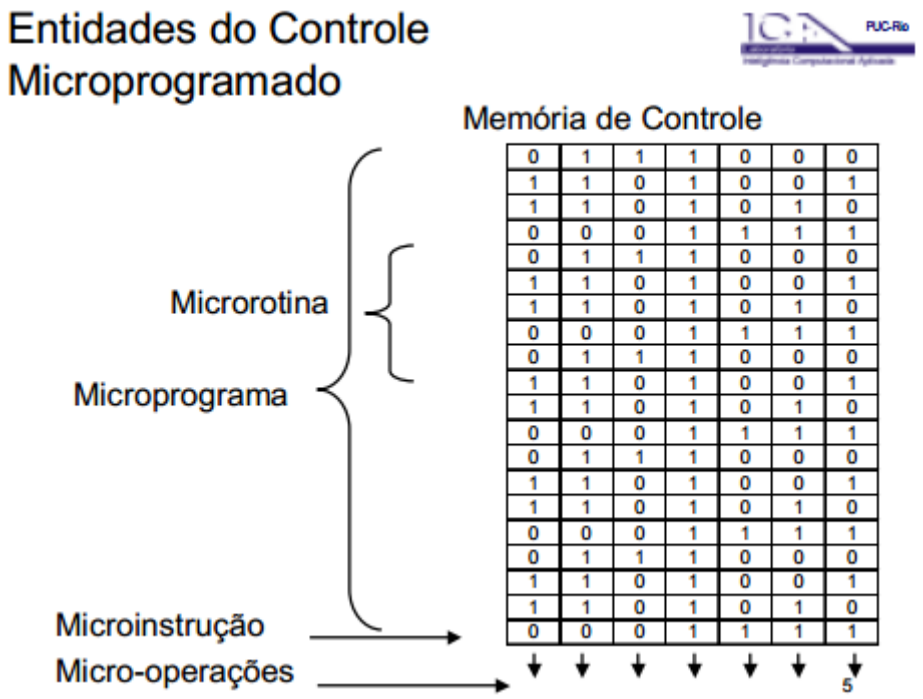

Figura 6: Entidades do Controle Microprogramado (**??**)

4.1.9.2 RISC - Reduced Instruction Set Computer

Na década de 1980, ocorreu as mudanças para a nova arquitetura, o modelo RISC (Reduced Instruction Set Computer). As melhorias nas linguagens de programação, tecnologia de compiladores e custo de memória significaram que, menos programação estava sendo feita no nível do assembly, de modo que os conjuntos de instruções poderiam ser medidos pela forma como os compiladores os usavam, ao contrário de como os programadores em assembly os usavam. Praticamente todos os novos conjuntos de instruções desde 1982, seguiram essa filosofia RISC de tamanhos de instrução fixos, conjuntos de instrução load/store, modos de endereçamento limitados e operações limitadas. ARM, Hitachi SH, IBM PowerPC, MIPS e Sun SPARC são todos exemplos de arquiteturas RISCs (**??**).

4.1.9.3 Comparação entre RISC e CISC

Como visto anteriormente, a arquitetura CISC apresenta instruções complexas executadas em vários ciclos de clock, enquanto a arquitetura RISC possui somente instruções que são executadas em apenas um ciclo. No quesito de acesso a memória, o conjunto complexo possui vários tipos de modos de endereçamento de memória, facilitando o trabalho do programador. Os microprocessadores RISC são considerados máquinas *load/store*, o que é possível pois ele possui uma grande quantidade de registradores dos mais variados tipos. A clara vantagem da arquitetura RISC é em questão de velocidade, pois por possuir um conjunto de instruções, com todas instruções com formato fixo, ocorre um uso intenso de *pipeline*. No desenvolvimento de um microprocessador CISC, a complexidade do sistema se encontra no microprograma, como visto um exemplo na figura 6, e na arquitetura RISC a complexidade se encontra no compilador. Ambas arquiteturas são muito bem aceitas no mercado e cada uma possui suas vantagens e desvantagens para serem aplicados em diversos tipos de projetos.

4.1.10 Memória Cache

De acordo com (**??**), a memória cache consegue realizar a ponte entre a diferença de velocidade entre o processador e a memória. A cache é um pequeno espaço de alta velocidade que se situa entre o processador e a memória na hierarquia de memórias.

Cache, foi o nome escolhido para representar o nível na hierarquia de memória entre o processador e a memória do primeiro computador comercial a ter este nível extra, como pode ser visto na Figura 7 item (b). A razão da cache ($SRAM$) ser menor é devido a maiores decodificadores de endereço, pois são mais lentos do que menores decodificadores de endereço. Quanto maior a memória é, mais complexo é seu decodificador de endereço, e mais tempo leva para identificar o valor da posição de memória do endereço desejado (**??**).

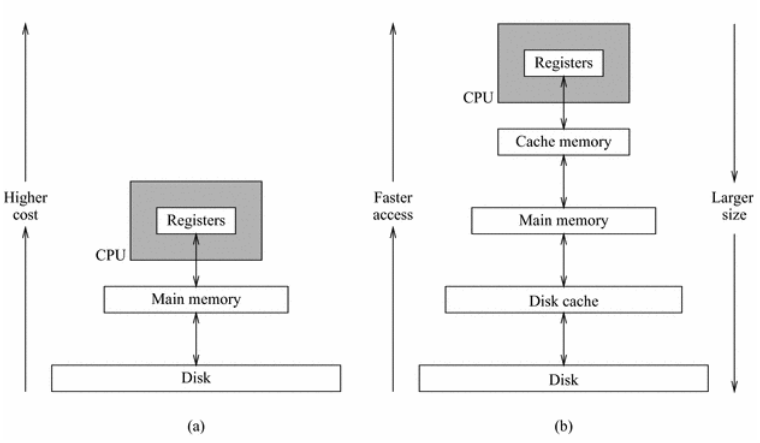

Figura 7: Hierarquia de Memórias (**??**)

É possível utilizar este conceito e dar um passo a diante introduzindo uma *SRAM* menor entre o cache o e processador, dentro do próprio envólucro do processador, criando dois níveis de memória cache L1 e L2, como podemos ver na figura 8 (**??**).

Figura 8: Dois níveis de memória cache L1 e L2 (**??**)

4.1.11 Pipeline

4.1.11.1 Definição

De acordo com (**??**),*Pipelining* é uma técnica de implementação no qual multiplas instruções são sobrepostas durante a execução, como visto na figura 9. Hoje em dia, *pipelining* é a chave para fazer processadores rápidos (**??**).

Como podemos ver, a utilização de *pipeline* torna a execução muito mais rápida do que se as tarefas fossem executadas sequencialmente. Como exemplo utilizare-

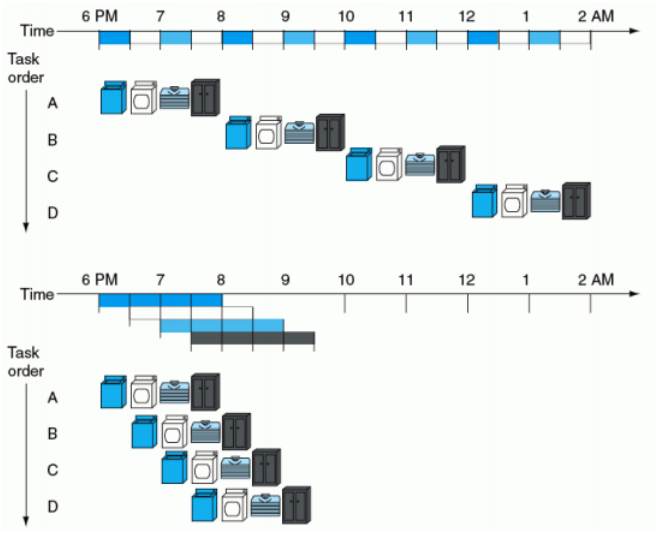

Figura 9: A analogia de uma lavanderia com o pipeline (**??**)

mos o microprocessador MIPS, que possui 5 estágios de execução de uma instrução, sendo elas: Decodificação da Instrução (*Instruction Fetch)*, Leitura dos Registros (*Reg*),Operação de ULA (*ALU*) , Acesso ao dado e Escrita no Registro. Na figura 10 podemos ver quantitativamente a diferença do processo com utilização de *pipeline*.

4.1.11.2 Desenvolvimento de um conjunto de instrução para o Pipeline

Primeiramente, todas as instruções do MIPS possuem o mesmo comprimento, esta restrição faz com que fique mais fácil decodificar as instruções no primeiro estágio e n segundo estágio do *pipeline*. Em um conjunto de instruções, como o do IA-32, onde instruções variam entre 1 byte até 17 bytes, o *pipelining* é consideravelmente mais complicado (**??**). Atualmente a arquitetura do IA-32 transforma as instruções em microinstruções, sendo essas realmente que serão utilizadas para a realização do *pipeline* com uma arquitetura *CISC*.

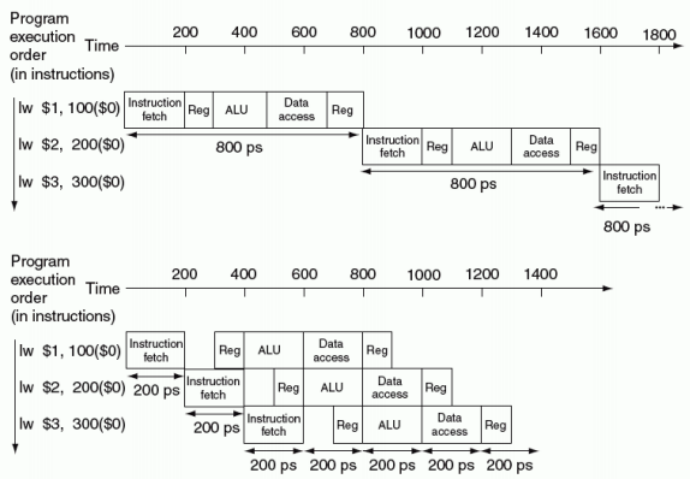

Figura 10: Comparação quantitativa da utilização de *Pipeline* utilizando a instrução *Load Word* do microprocessador MIPS. (**??**)

4.1.11.3 Problemas doPipeline

O **primeiro** tipo de problema do *pipeline*, é o problema estrutural, que significa que o hardware não suporta a combinação de instruções ao qual desejamos que sejam executadas em um único ciclo de clock. Ao caso de termos somente uma única memória, no segundo e quarto passo, são necessários acessos a memória que não podem ser executadas de uma única vez (**??**). O **segundo** tipo de problema , quanto a acesso aos dados, ocorre quando o *pipeline* fica travado no momento em que ocorre uma etapa deve esperar outra etapa completar para continuar, por exemplo no seguinte trecho de código do MIPS, na figura 11.

```
add    $s0, $t0, $t1
sub    $t2, $s0, $t3
```

Figura 11: Trecho de código que exemplifica um problema do *pipeline* (**??**).

Quando este tipo de problema ocorre, um fato chamado *bubble* aparece no meio do *pipeline*, como se fosse uma linha vazia entre dois processos do *pipeline*, podendo ser notado na figura 12.

Para resolver este tipo de problema, o código pode ser reescrito de uma forma

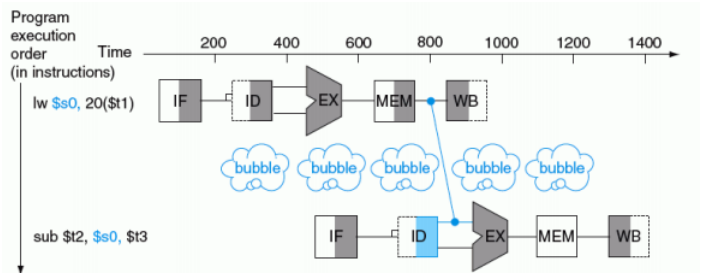

Figura 12: Fenômeno do tipo *bubble* no *pipeline* semelhante ao problema da figura 11 (**??**).

que evite a dependência das informações, essa correção pode ser realizada tanto pelo compilador como pelo programador.

O terceiro tipo de problema, é chamado de problema de controle, também chamado de problema de *branch*, surge da necessidade de tomar uma decisão baseada nos resultados de uma instrução enquanto outras estão em execução (**??**).

As instruções de *branch* são instruções de desvios condicionais no código, portanto a execução da próxima instrução depende se o *branch* causará desvio ou não, caso o desvio ocorra, o fenômeno de *bubble* ocorre novamente, pois o *pipeline* necessita ficar um tempo parado esperando a tomada de decisão, semelhante a figura 13.

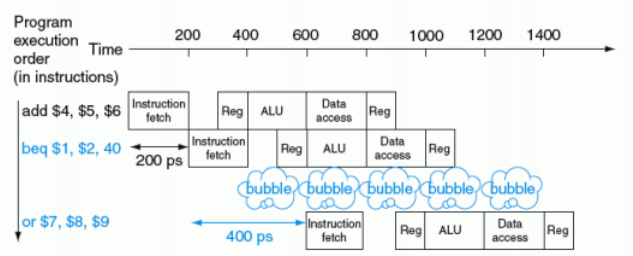

Figura 13: Fenômeno do tipo *bubble* no *pipeline* quando ocorre o problema de *branch* (**??**).

Para solucionar este problema, desenvolveu-se uma estrutura dentro do próprio microprocessador, chamada *branch predictor*. O *branch predictor*, é um circuito digital que tenta adivinhar qual vai ser o caminho que o branch irá seguir, para

continuar preenchendo o *pipeline*. Hoje em dia, tal estrutura desempenha um papel fundamental para o desenvolvimento de processadores de alta performance.

4.1.12 Processadores Multi-Core e Hyper-Threading

De acordo com (**??**), basicamente, multi-core é um design ao qual um único processador físico, contém o núcleo lógico de mais de um processador, como pode ser visto na figura 14. O objetivo deste design é habilitar o sistema a executar mais tarefas simultaneamente e desse modo alcançar uma maior performance do sistema.

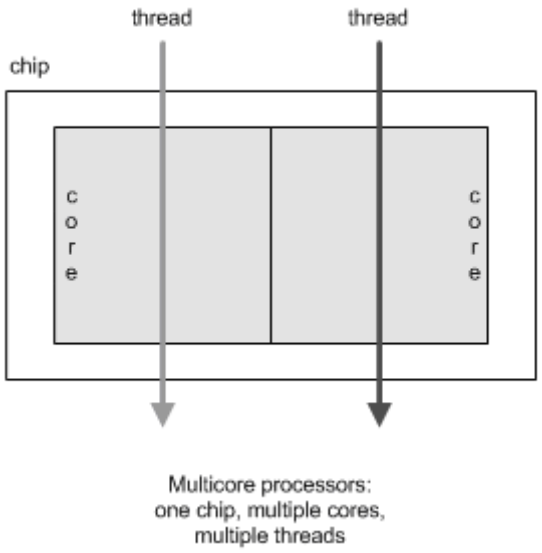

Figura 14: Processador com dois núcleos dentro de um único processador (**??**).

Programas são executados, à partir de threads, essas threads são sequências de instruções relacionadas. Nos primórdios do PC, maioria dos programas consistia de uma única thread, o sistema operacional naquela época era capaz de executar somente um programa por vez, tendo como resultado uma sensação dolorosa que seu PC congelava enquanto imprimia um documento ou uma folha de trabalho, o sistema era incapaz de realizar duas tarefas simultaneamente. Inovações no sistema

operacional introduziram os sistemas multitarefa, no qual um programa pode ser brevemente suspendido enquanto executa outra, de uma maneira que o usuário não perceba, pealizando esta troca rapidamente, o sistema tem a aparência de estar executando os programas simultaneamente, contudo o processador estava de fato, o tempo executando uma única thread.

No início dos anos 2000, o design de processadores ganhou recursos adicionais, como uma lógica dedicada para operações com ponto flutuante, para suportar a execução de múltiplas instruções em paralelo. A Intel®, definiu que o melhor uso desses recursos empregando-as para executar duas threads simultaneamente no mesmo núcleo de processamento, figura 15. A Intel®nomeou este procedimento simultâneo como *Hyper-Threading Technology*®e lançou-a nos processadores **Intel Xeon ®** em 2003. De acordo com medidores da Intel®, aplicações que eram escritas utilizando múltiplas threads realizaram suas tarefas 30% mais rápido do que se for executado utilizando a tecnologia **HT**. Para induzir o sistema operacional a reconhecer um processador como duas possibilidades de execução de *pipeline*, *chips* foram feitos para aparentar ser dois processadores lógicos.

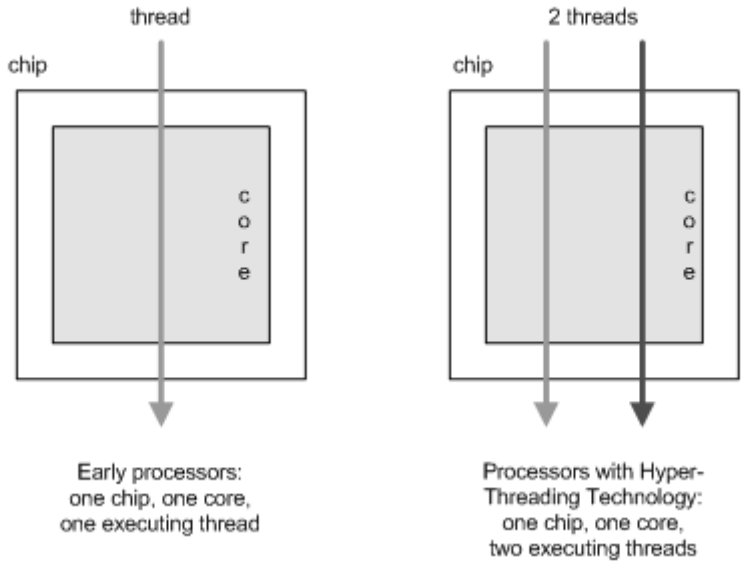

Figura 15: Exemplo de um processador com a tecnologia *Hyper-Threading* (**??**).

4.2 Microprocessador 8086/8088

4.2.1 História

Em 1968 a empresa Intel foi fundada por Robert N. Noyce, Gordon E. Moore e Andrew Grove. Robert N. Noyce foi o inventor do circuito integrado.

Em 15 de novembro de 1971 nascia o processador 4004 de apenas 4 bits e grande capacidade para realizar operações aritméticas. Esse microprocessador possuía 2.300 transistores para processar 0,06 milhões de instruções (60.000) por segundo e não tinha o tamanho de um selo de carta. Para se ter uma idéia, o ENIAC, primeiro computador de que se tem notícia , construído em 1946 para fins bélicos, ocupava sozinho 1.000 metros quadrados e fazia o mesmo que o 4004.

O 4004 foi usado apenas para cálculos poucos complexos (4 operações), ele era um pouco mais lento que Eniac II mais tinha a vantagem de possuir a metade do tamanho, esquentar menos e consumir menos energia.

Surgiu em 1972 o 8008, primeiro processador de 8 bits, com capacidade de memória de 16 Kbytes (16.000 bytes), enquanto o 4004 possuía apenas 640 bytes.

Em 1974 é lançado o 8080, com desempenho seis vezes maior que o anterior com um clock de 2MHz, rodava um programa da Microsoft chamado Basic, possuía apenas led's. Além de 16Kb de memória Rom onde ficava o sistema, possuía 4Kb de memória Ram, seus controles eram através de botões, possuía drive de disquete 8"com capacidade de 250 Kb.

O 8086 foi o primeiro processador feito pela Intel para ser usado com os PC's. Ele contava com um barramento de dados interno e externo de 16 bits. E foi este o motivo de não ter sido o processador mais utilizado. Inicialmente ele foi distribuído em versões de 4,77MHZ. Posteriormente vieram versões turbinadas de 8 e 10 MHZ.

A história do 8086 é bem simples. Quando ele foi lançado, a maioria dos

dispositivos e circuitos disponíveis eram de 8 bits. Era muito caro adaptar todo o resto do computador por causa do processador. E foi isso que acabou com o 8086. Para adaptar-se a este mercado a Intel lançou o 8088, com barramento externo mais lento, de 8 bits. Deixando a diferença de barramento externo, ambos eram idênticos.

Quando este chip, o 8086, veio a ser utilizado já era tarde demais. Ele chegou até a fazer parte de uns poucos clones do IBM PC e posteriormente em dois modelos do IBM PS/2 e de um computador Compaq. Mas sua destruição veio com um processador mais poderoso, o 80286.

Outro possível fator para a pouca aceitação deste processador pode ter sido a falta de unidades devido à demanda. Nunca havia chips suficientes para produzir computadores em grande escala.

4.2.2 Visão preliminar

Tanto o 8086 como o 8088 utilizam o conceito de fila de instruções para melhorar a velocidade do computador. Uma área no interior da pastilha denominada fila de instruções retém diversos bytes de uma instrução. Quando o computador estiver pronto para a próxima instrução, ele não precisa pegar muitos bytes na memória, uma vez que toda instrução poderá já se encontrar na fila. O conceito de fila aumenta o numero de operações realizadas por segundo uma vez que o processador vai estar utilizando o bus de dados e endereços por menor período de tempo, disponibilizando este para outros dispositivos. A fila do 8086 tem 6 bytes de largura e a do 8088 tem 4 bytes.

O 8086 pode acessar 1 megabyte de memória de leitura escrita (220 bytes). Entretanto ele utiliza um esquema de endereçamento de memória denominado segmentação, em que determinados registradores de segmento fornecem um endereço básico que é automaticamente acrescentado a cada endereço a cada endereço de usuário de 16 bits na máquina.

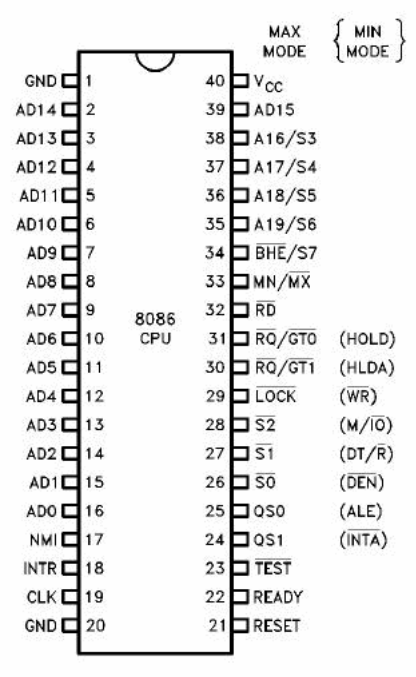

Figura 16: Pinagem do IA-PX 86

A parte do endereço e todas as vias de dados são multiplexados em 16 pinos (os 16 pinos do barramento de dados é o que o classifica como um microprocessador de 16 bits). Os restantes 4 bits de endereço são implementados por quatro pinos adicionais de endereço, que também são utilizados para status (como mostra a figura 1). È requerido um clock externo á pastilha e é utilizado um controlador de via externo à pastilha para demultiplexar a via de dados e de endereço.

O 8086 tem uma estrutura de interrupção poderosa. Quase todos os microprocessadores de 8 bits requerem pastilhas externas adicionais para permitir operações de interrupção adequadas. No 8086, cerca de 1000 bytes são colocados de lado para conter até 265 apontadores de vetores (lembrando que cada apontador é um endereço contendo offset:seletor, ou seja, 4 bytes). O 8086 executa operações de E/S (ou I/O) em um espaço separado da memória denominado espaço de E/S. Tem um total de 64k bytes. Para ser utilizado pelos co-processadores é fornecido um pino de entrada TEST especial (pino 23, figura 1) para permitir ao 8086 saber

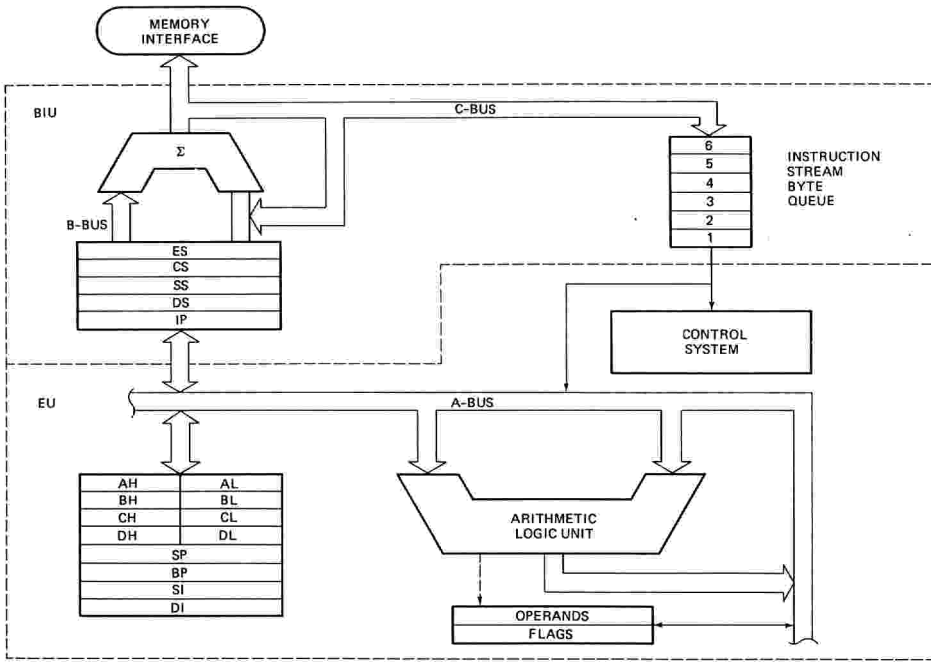

Figura 17: Arquitetura do 8086/8088

quando é que o co-processador completou a tarefa. Quando uma instrução WAIT é acionada, o 8086 pára e aguarda do co-processador externo, ou qualquer outro hardware, um sinal para que continue pela alteração do pino TEST.

4.2.3 Memória

A largura de memória "vista"por um microprocessador é ditada pela quantidade de bits que o microprocessador pode acessar por vez. Esta quantidade é determinada pela largura do seu barramento de dados externo. O microprocessador 8086 possui um barramento de dados de 16 bits o que permite acessar dois bytes da memória consecutivamente (cada referência à memória acessa 2 bytes), enquanto, o 8088 possui um barramento de dados de 8 bits, ou seja, uma referência à memória acessa 1 byte.

Os microprocessadores 8086/8088 podem acessar (ler ou escrever) bytes ou

palavras (2 bytes) que se localizam tanto em endereços pares como em endereços impares da memória. No entanto, dependendo do microprocessador e do tamanha do dado a ser acessado, pode ser necessária que o microprocessador efetue uma ou duas referências para a memória. A tabela 1 resume o número de referencias necessárias para os microprocessadores 8086/8088 acessarem dados de 8 bits e 16 bits em endereços pares e impares da memória.

Tamanho do dado	Endereço	Numero de referências	
		8086	8088
Byte	Par	1	1
	Impar	1	1
Palavra	Par	1	2
	Impar	2	2

O espaço de endereçamento da memória do microprocessador 8088 é organizado como um vetor linear de 1 MByte, sendo que cada localização deste espaço é referenciada por meio de um endereço único de 20 bites chamado endereço físico.

No microprocessador 8086 o espaço de endereçamento da memória é organizado em dois bancos de 512KBytes cada, chamados bancos par e banco impar, respectivamente. O banco par é conectado ao 8086 por meio de 8 bits menos significativos do barramento de dados, já o banco par é conectado por meio dos 8 bits mais significativos do barramento de dados.

4.2.4 Arquitetura do microprocessador

A unidade de execução (EU) executa as operações aritméticas e lógicas, além de controlar a maioria dos registros internos e manipular os dados. Em contraste, a unidade de interface de barramento (BIU representado por um símbolo de somatória na figura 1) executa as operações de barramento, incluindo a transferência

de dados, e controla os registros restantes do microprocessador.

As duas unidades de processamento são capazes de executar suas operações de forma independente, isto é, cada unidade pode fazer suas tarefas sem a assistência da outra unidade.

Embora a unidade de execução EU esteja isolada do barramento do sistema, pela unidade de interface de barramento (BIU), ela ainda pode acessar o barramento para certas operações. A EU ganha o acesso do barramento requisitando que a BIU temporariamente suspensa suas atividade. A EU então assume o controle da BIU de modo a poder usar o barramento para enviar ou receber informações.

Da figura 1 nota-se que a EU consiste de duas seções principais que são os registros de propósitos gerais e a unidade aritmética e lógica - ALU. Os registro de propósitos gerais do 8086/8088 presentes na EU são áreas de armazenamento com capacidade de manter dados binários que foram ou serão usados pelo microprocessador. A grande vantagem destes registros se deve ao fato de se poder acessá-los com maior facilidade e de forma muito mais rápida do que uma determinada localização da memória.

Todos registros de propósitos gerais possuem capacidades aritméticas e lógicas e dados podem ser armazenados através da BIU ou transferidos para memória. Os registros de propósitos gerais são todos de 16 bits, entretanto os bytes menos e mais significativos podem ser usados separadamente como registros de 1 byte, dessa forma tem-se os seguintes registros AH, AL, BH, BL, CH, CL, DH e DL.

A maioria das operações que se pode executar em um dos registros de propósitos gerais podem também serem executadas nos demais registros. Contudo, os registros de propósitos gerais tem uso especifico para algumas poucas instruções, ou seja, algumas instruções utilizam de forma específica determinados registros. Devido a este fato os registros de propósitos gerais recebem nomes descritivos que são: Acumulador (AX), Base (BX), Contador (CX), Dado (DX), índice fonte (SI), índice destino (DI), ponteiro base (BP) e ponteiro da pilha (SP).

A unidade aritmética lógica, ALU, recebe instruções e então executa sobre os dados especificados pela instrução uma operação aritmética, como soma ou subtração ou lógica como OR ou AND.

A seção de lógica de controle de barramento é responsável por todas as operações de barramento do microprocessador como, por exemplo, a busca de dados para a unidade aritmética lógica (ALU). Quando necessário a seção de lógica de controle de barramento acessa localizações particulares na memória, de modo que a EU possa enviar ou receber informações para ou destas localizações.

A seção de lógica de controle de barramento também controla o sentido do fluxo de informação no barramento. Quando uma informação tiver que ser enviada à memória, esta seção assegura que os sinais de controle sejam os apropriados para a transmissão. O mesmo ocorre quando for necessário receber uma informação. A fila de instruções age como um "encanamento"onde os bytes das instruções trazidos da memória são armazenados antes do seu uso pela EU. No 8086 esta fila é composta de 6 localizações de 8 bits cada, enquanto no 8088 a fila de instruções é composta de 4 localizações de 8 bits. Pode-se dizer que estas localizações servem como áreas para o armazenamento temporário (buffer) das instruções trazidas da memória.

No microprocessador 8086/8088 é a seção lógica de controle de barramento BIU, que busca os bytes das instruções do programa na memória e os coloca na fila de instruções. Esta fila mantém estes bytes até que a EU esteja pronta para aceitá-las.

Devido as características do microprocessador 8086, a BIU sempre busca as instruções acessando palavras (16 bits) que se encontram armazenadas em endereços pares. A única exceção ocorre quando existe um desvio (JUMP) para uma instrução que se encontra armazenada na memória em um endereço impar. Quando isso ocorre o 8086 traz para a fila de instruções um único byte da instrução e a seguir continua acessando palavras que se encontram armazenadas em endereços pares. Este fato não ocorre em um 8088 visto que seu barramento de dados é de 8 bits. É importante salientar que as instruções de um microprocessador 8086 poder

ter de um a seis bytes de comprimento.

Independente do microprocessador, caso ocorra um desvio na seqüência de execução das instruções, a fila de instruções é automaticamente esvaziada e a BIU passa a buscar as instruções a partir da nova localização de memória para o qual se deu o desvio.

4.2.5 Endereçamento da memória

Ao contrário dos microprocessadores que utilizam um modelo de memória lineal, ou seja, que enxergam o seu espaço de endereçamento de memória de forma lineal, o 8086/8088 utiliza um modelo de memória denominado segmentada. Neste modelo o microprocessador enxerga o espaço de endereçamento de memória dividido em vários segmentos.

Um segmento nada mais é do que uma região continua do espaço de endereçamento de memória que é tratada pelo microprocessador como uma unidade lógica. Por serem unidades lógicas e não físicas, os segmentos podem localizar-se em qualquer parte do espaço de endereçamento linear. Conseqüentemente, dois ou mais segmentos distintos poder ser: adjacentes, parcialmente sobrepostos, totalmente sobrepostos ou desconexos.

No modelo de memória segmenta do 8086/8088, o microprocessador somente pode acessar as localizações de seu espaço de endereçamento por meio de um determinado segmento. Assim para este microprocessador, um segmento funciona como uma ”janela móvel”sobre o seu espaço de endereçamento lineal, através do qual ele acessa as localizações do seu espaço de endereçamento.

Para o microprocessador 8086/8088, os segmentos podem se localizar em qualquer parto do seu espaço de endereçamento de memória. Entretanto, devido a arquitetura deste microprocessador, um seguimento somente pode começar em uma localização do espaço de endereçamento de memória, cujo endereço físico seja múltiplo de 16 (10H).

O endereço físico do inicio de um segmento do 8086/8088 é designado por endereço base, os 16 bits mais significativos do endereço base corresponde a um endereço chamado endereço de segmento ou seletor. Conseqüentemente, cada segmento do 8086/8088 é identificado por um endereço de segmento ou seletor de 16 bits.

Dentro de cada segmento o endereçamento se da de forma linear, porem relativo ao inicio do endereço do segmento. Cada localização do espaço de endereçamento de memória dentro do segmento é identificado por meio de um endereço de 16 bits chamado endereço efetivo (Effective Address - EA) ou endereço de offset (offset).

Devido a segmentação do espaço de endereçamento de memória e ao endereçamento relativo dentro do segmento, cada localização do espaço de endereçamento de memória do microprocessador é identificado por meio de um endereço de 32bits chamado de endereço lógico (seletor : offset).

A forma como o microprocessador converte um endereço lógico de 32 bits em um endereço físico de 20 bits, faz com que vários endereços lógicos identifiquem uma mesma localização do espaço de endereçamento de memória. A conversão de endereço lógico em endereço físico se da multiplicando o seletor por 10h e em seguida somando o offset. O endereço físico pode ser representado na forma normalizada para obter-se o endereço lógico, os 4 bits menos significativos do endereço físico correspondem ao endereço de offset e os 16 bits mais significativos do endereço físico correspondem ao endereço de segmento.

4.2.6 Conjunto de registros

Embora os registros SI, DI, BP e SP, possuam capacidade aritmética e lógica de 16bits, como os demais registros de propósitos gerais de 16 bits, estes registros geralmente são usados para manter o endereço efetivo (offset) de localizações de memória ou para apontar estruturas de dados na memória.

Particularmente, o registro SP é utilizado para manter o endereço efetivo do

topo de uma estrutura de dados na memória que funciona como uma pilha (stack), onde o último dado a ser armazenado nesta estrutura deverá ser o primeiro a ser retirado (LIFO - Last Imput/first output). Uma vez que esta estrutura é de vital importância para o funcionamento do microprocessador, a utilização do registro SP em operações aritméticas ou lógicas não é aconselhada.

Como o microprocessador 8086/8088 utiliza um modelo de memória segmenta, o acesso as localizações do seu espaço de endereçamento de memória é feito através de segmentos mediante endereços lógicos de 32bits. Por isso, para poder acessar as localizações dentro de um determinado segmento é necessário que o 8086/8088 conheça o endereço deste segmento, ou seja, o seu seletor. Para isso, o microprocessador 8086/8088 dispõe de um conjunto de registros especiais chamados registro de segmentos, cuja finalidade, como o próprio indica, é mates endereços de segmentos.

Para acessar uma localização do espaço de endereçamento de memória que não é abrangido por um dos segmentos apontados pelos registros de segmentos, é necessário alterar o conteúdo de um dos registros de segmento, de modo que este registro aponte para um segmento que venha a abranger a localização que se deseja acessar.

Um programa, geralmente, é composto por três partes ou segmentos que são: o segmento de códigos, o segmento de dados e o segmento de pilha. As partes ou segmentos de um programa podem residir em qualquer ordem e em qualquer lugar do espaço de endereçamento de memória que tenha memória física.

Parte do programa	Registro de segmento
Código do programa	CS
Dados do programa	DS
Pilha do programa	SS
Área extra da memória	ES

O ponteiro de instruções (IP) é um registro de 16 bits, que sempre matem o endereço efetivo da localização de memória onde está armazenado o código de máquina da próxima instrução a ser executada. Este registro é automaticamente incrementado pelo microprocessador de acordo com o tamanho desta instrução.

A representação de um endereço lógico se dá na forma seletor:offset. Quando os conteúdos de dois registros são usados para especificar um endereço lógico, o endereço lógico geralmente é escrito na forma RS:RO onde RS corresponde ao nome do registro que mantém a parte do endereço lógico que se refere ao endereço de segmento e RO corresponde ao nome do registro que mantém a parte do endereço lógico que se refere ao endereço efetivo (offset). No caso os registros que podem ser utilizados para apontar localização dentro do segmento de dados são os registros BX, SI e DI.

No segmento de stack o ponteiro de stack (SP) mantém o endereço efetivo da localização de memória que corresponde ao topo da pilha. O endereço de segmento é mantido no registro de segmento SS.

O poder real de um microprocessador está na sua capacidade de tomar decisões. O 8086/8088 baseia as suas decisões no conteúdo de um registro de 16 bits chamado registro de flags. Este registro é automaticamente atualizado para manter informações a respeito da ultima operação aritmética ou lógica que o microprocessador executou. Apenas 9 flags do registro de flags são utilizados, são eles: OF - overflow, DF - direção, IF - interrupção, TF - armadilha, SF- sinal, ZF - zero, AF - carry auxiliar, PF- paridade e CD - carry.

4.2.7 Instruções

Cada instrução (cartão de referencia do microprocessador 8086/8088 em anexo) possui uma representação binária única que é conhecida por código de máquina. Este modelo binário ou código de máquina da instrução, quando for aplicado aos circuitos internos do microprocessador faz com que ele execute uma

operação particular. Uma instrução de um modo geral pode ser dividida em duas partes: a do código de operação (opcode) e a dos seus operandos.

O código de operação (opcode) é a parte da instrução que identifica a operação básica a ser executada pelo microprocessador. Enquanto, os operandos identificam os dados que devem ser utilizados pelo microprocessador na operação identificada pelo código de operação.

Dependendo da instrução ela pode ter 2 operandos, 1 operando ou nenhum operando. Na representação das instruções com dois operandos, o operando destino é sempre especificado em primeiro, e este é separado do operando fonte por uma vírgula. Quando um operando se refere a um registro de 8 ou 16 bits ele é chamado de operando registro, e quando ele refere a uma localização de memória ele é chamado operando memória. Por outro lado, um operando imediato se refere a um dado de 8 ou 16 bits que é especificado na própria instrução, ou seja, é especificado no código de máquina da própria instrução.

Um operando pode se encontrar em um registro (operando registro), numa localização de memória (operando memória), num dispositivo periférico de entrada/saída, ou até mesmo estar codificado no próprio código de máquina da instrução (operando imediato). A maneira na qual a localização de um operando é especificada chama-se modo de endereçamento. O 8086/8088 utiliza duas categorias de endereçamento geral que são, o modo de endereçamento registro ou modo registro e o modo de endereçamento memória ou modo memória. Pode-se dizer que uma instrução do 8086/8088 utiliza o modo de endereçamento registro quando nenhum dos operandos da instrução se refere a memória. Por outro lado, uma instrução utiliza o modo de endereçamento memória quando um dos seus operandos da instrução se refere a um operando memória.

Quando um operando se refere a um operando memória até 3 valores de 16 bits podem ser somados para especificar o endereço efetivo (offset) do operando memória. Nesta soma qualquer carry que venha a ocorre é ignorado pelo microprocessador, por um endereço efetivo (offset) no 8086/8088 é representado por um

número de 16 bits.

Sobre as 2 categorias gerais de endereçamento existem 7 modos específicos de endereçamento.

4.2.7.1 Endereçamento por registro

Uma instrução utiliza o modo de endereçamento por registro quando o operando destino e o operando fonte desta instrução forem ambos operando registros.

Exemplo: MOV AX, BX.

4.2.7.2 Endereçamento imediato

Uma instrução utiliza o modo de endereçamento por registro quando o operando fonte desta instrução dor imediato.

Exemplos: MOV CX, 1234h (modo registro); MOV [2011H], 1234H (modo memória).

4.2.7.3 Endereçamento Direto

Uma instrução utiliza o modo de endereçamento direto quando o operando destino ou operando fonte da instrução se refere a uma localização de memória, cujo endereço efetivo é especificado na própria instrução.

Exemplos: MOV CX, [1234H]; MOV [1234H], DX

4.2.7.4 Endereçamento indireto por registro

Uma instrução utiliza o modo de endereçamento indireto por registro quando o operando destino ou o operando fonte da instrução se refere a um operando memória, cujo endereço efetivo se encontra armazenado num registro.

Exemplo: MOV CX, [BX];

4.2.7.5 Endereçamento por base

Uma instrução utiliza o modo de endereçamento por base quando o operando destino ou operando fonte da instrução se refere a um operando memória, cujo endereço efetivo (EA) é especificado pela soma do conteúdo do registro BX ou BP com um número de 8 ou 16 bits chamado deslocamento. No caso do deslocamento de 8 bits o microprocessador estende o sinal até se obter um numero binário sinalizado de 16 bits, quando ocorrer um deslocamento de 16 bits o microprocessador interpreta como um numero absoluto de 16 bits.

Quando o conteúdo do registro BP é utilizado no calculo, o 8086/8088 automaticamente associa o endereço efetivo do operando memória com o conteúdo do registro de segmento de stack (registro SS), de modo a formar o endereço lógico do operando memória. Neste caso, portando, o 8086/8088 acesa o operando memória no segmento da pilha.

Exemplo: MOV AX, [BX + 1000H]

4.2.7.6 Endereçamento Indexado

Uma instrução utiliza o modo de endereçamento indexado quando o operando destino ou o operando fonte da instrução se refere a um operando memória, cujo endereço efetivo é especificado pela soma do conteúdo do registro SI ou DI, com um número binário de 8 ou 16 bits chamado deslocamento. No caso do deslocamento de 8 bits o microprocessador estende o sinal até se obter um numero binário sinalizado de 16 bits, quando ocorrer um deslocamento de 16 bits o microprocessador interpreta como um numero absoluto de 16 bits.

Exemplo: MOV AX, [SI + 2000H]

4.2.7.7 Endereçamento por base indexado

Uma instrução utiliza o modo de endereçamento por base indexado quando o operando destino ou o operando fonte da instrução se refere a um operando memória, cujo endereço efetivo é especificado pela soma do conteúdo do registro BX ou BP, com o conteúdo do registro SI ou DI e opcionalmente um número binário de 8 ou 16 bits chamado deslocamento.

Exemplo: MOV AX, [BX + SI + 2000H]

4.3 VHDL

A linguagem VHDL foi desenvolvida pela necessidade de um padrão para o intercâmbio de informações entre fornecedores de equipamentos para o Departamento de Defesa dos Estados Unidos. Esta linguagem é usada para descrever o comportamento de circuitos ou sistemas eletrônicos a partir de um sistema físico. É importante lembrar que esta é uma linguagem concorrente, ou seja, os comandos envolvidos em um mesmo evento acontecem simultaneamente, diferentemente de linguagens de programação de software. Além disso, é uma linguagem portável, ou seja, independe da tecnologia ou do fornecedor. A Figura 1 mostra as etapas de um projeto utilizando VHDL.

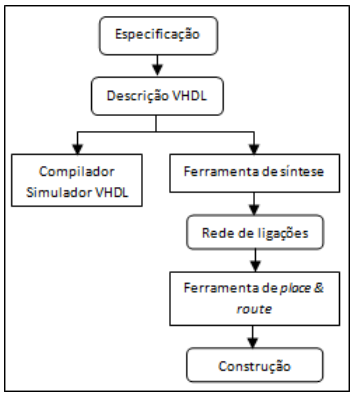

Figura 18: Etapas de Projeto Usando VHDL

Na lógica programável, há dois dispositivos principais: CPLD (*Complex Programmable Logic Devices*) e FPGA (*Field Programmable Gate Arrays*) no campo de ASIC (*Application Specific Integrated Circuits*). A partir do código VHDL, pode-se fabricar um chip de alta complexidade ou executá-lo em um dispositivo programável.

O código VHDL é composto de três partes principais: Biblioteca, Entidade e Arquitetura.

1. Biblioteca (*Library*) : É composta de todas as bibliotecas usadas no projeto.
2. Entidade (*Entity*): Determina as entradas e saídas do circuito.
3. Arquitetura (*Architeture*): Contém o código VHDL que descreve a forma de como o circuito deve ser comportar (*function*).

4.3.1 Biblioteca

Uma biblioteca têm várias implementações de código que são úteis a outros projetos. A Figura 19 ilustra a estrutura típica de uma biblioteca. O código, normalmente, é escrito na forma de funções (*Functions*), procedimentos (*Procedures*) ou componentes (*Components*), que ficam dentro de pacotes (*Packages*) e depois é compilado na biblioteca.

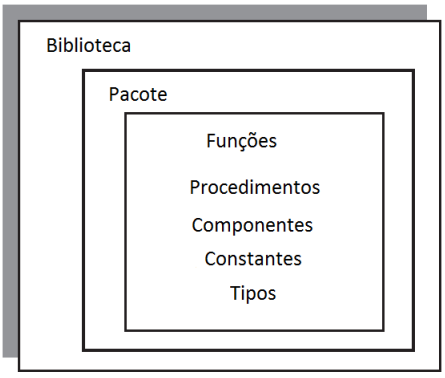

Figura 19: Estrutura de uma Library (**??**)

4.3.2 Entidade

Uma entidade de projeto pode representar uma simples porta lógica como um sistema completo. A declaração da entidade define a interface com o ambiente exterior, como, por exemplo, as entradas e saídas. Os quatro modos de porta são:

1. IN : Apenas entrada.
2. OUT: Apenas saída.
3. BUFFER: Saída que controla sinal interno.
4. INOUT: Porta bidirecional

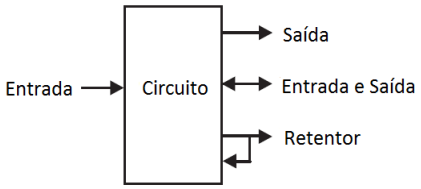

Figura 20: Tipos de Entrada e Saída (**??**)

4.3.3 Arquitetura

A arquitetura contém a parte lógica da entidade utilizando suas entradas e saídas. Ainda é possível declarar sinais internos dentro da arquitetura, estes sinais são chamados classes. São elas:

1. CONSTANT - Define um objeto com valor estático.
2. VARIABLE - São objetos que podem ter o seu valor alterado, e são usadas em regiões de código seqüencial.
3. SIGNAL - São objetos que podem ter o seu valor alterado, e são usadas em regiões de código concorrente ou seqüencial. É bom lembrar que a porta de uma entidade realiza a declaração de um sinal.

```
ARCHITETURE architeture_name OF entity_name IS
        [declarations]
BEGIN
        (code)
END architeture_name;
```

Figura 21: Sintaxe de uma Architeture

A arquitetura é composta de duas partes, uma para declarações, onde sinais e constantes são declarados e outra onde fica o código. Como no caso da entidade, o nome da arquitetura pode ser qualquer nome (exceto palavras reservadas), até mesmo o nome da entidade.

4.4 FPGA

Um dispositivo FPGA é um semicondutor utilizado amplamente no processamento de informações. Criado pela Xilinx Inc., teve seu lançamento no ano de 1985 como um dispositivo que poderia ser programado de acordo com as aplicações do usuário (programador).

O FPGA é composto basicamente por três tipos de componentes: blocos de entrada e saída (IOB), blocos lógicos configuráveis (CLB) e chaves de interconexão (Switch Matrix). Os blocos de entrada e saída (I/O) formam uma borda ao redor do dispositivo. Cada um desses blocos pode servir como entrada, saída ou acesso bi-direcional a outros pinos de I/O. Os blocos lógicos são dispostos de forma bidimensional, as chaves de interconexão são dispostas em formas de trilhas verticais e horizontais entre as linhas e as colunas dos blocos lógicos como é mostrado na figura.

1. CLB (Configuration Logical Blocks): São circuitos construídos pela reunião de Flip-Flops (entre 2 e 4) juntamente com lógica combinacional. A partir de CLBs é possível construir elementos funcionais lógicos.

2. IOB (Input/Output Block): Circuitos que realizam o interfaceamento das

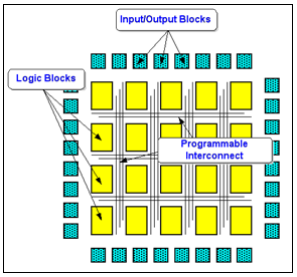

Figura 22: Estrutura de uma FPGA (**??**)

saídas provenientes das saídas das combinações de CLBs. São basicamente buffers, que funcionarão como um pino bidirecional entrada e saída do FPGA.

3. Switch Matrix (chaves de interconexões): Trilhas que conectam os CLBs e IOBs. O terceiro grupo é composto pelas interconexões. Os recursos de interconexões possuem trilhas para conectar as entradas e saídas dos CLBs e IOBs para as redes apropriadas. Geralmente, a configuração é estabelecida por programação interna das células de memória estática, que determinam funções lógicas e conexões internas implementadas no FPGA entre os CLBs e os IOBs. O processo de escolha das interconexões é chamado de roteamento.

4.4.1 Programação de FPGAs

De modo geral, a etapa inicial do projeto é seguida de uma simulação funcional. Neste ponto, é utilizado um simulador para conferir as saídas com as diversas entradas de teste. Uma vez de posse de uma representação funcional do hardware, a compilação é iniciada. Esta compilação é dividida em 2 etapas.

A primeira, chamada síntese (synthesys), é onde o compilador tem uma idéia de como implementar o projeto e falta apenas posicionar e rotear as estruturas lógicas em macrocélulas, interconexões e pinos de entrada e saída, que é a segunda etapa.

Feita a compilação, um bitstream é criado, ou seja, já sabemos quais são os dados binários que deverão ser carregados no FPGA ou CLPD para fazer com que

o chip execute um projeto em particular. Geralmente, as empresas que desenvolvem dispositivos programáveis, fornecem também softwares capazes de carregar o bitstream no hardware utilizando linguagens de descrição de hardware (Hardware Description Language ? HDL).

4.5 Arquitetura FPGA da família Cyclone II

Neste projeto será utilizado o kit de desenvolvimento DE1, Figura 24. Tal kit possui uma FPGA da família Cyclone® II que trabalha com o Quartus II cujas as características são apresentadas logo a seguir.

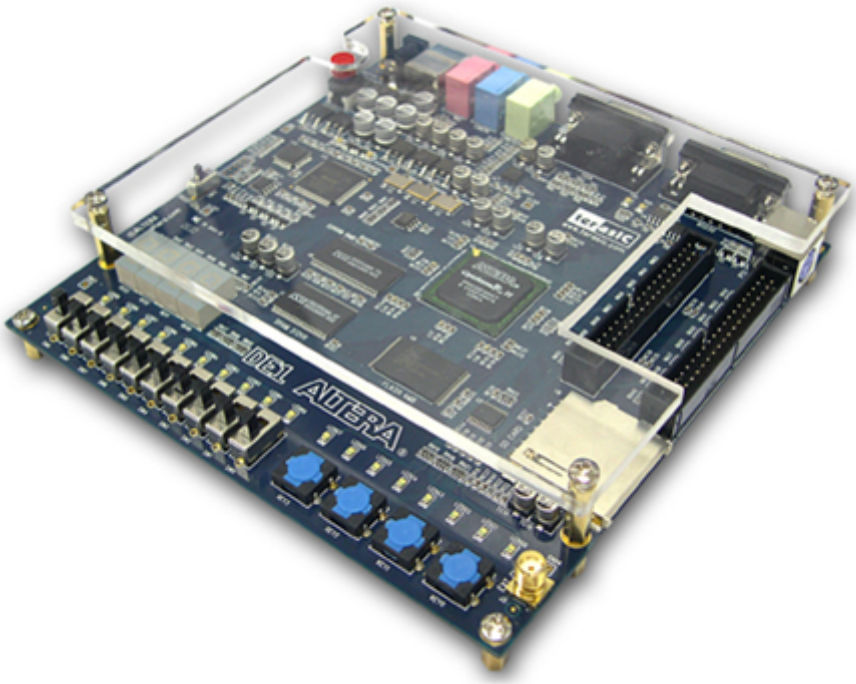

Figura 23: Kit de desenvolvimento DE1

A família de FPGAs Cyclone II® da Altera é manufaturada em wafers de silício de 300mm utilizando processadores com low-k dielétricos de 90nm da TSMC para garantir a rápida disponibilidade e o baixo custo. Minimizando a área de

silício e oferecendo suporte a sistemas digitais complexos em um único chip, estes dispositivos podem facilmente competir no mercado com os ASICs.

Tendo uma densidade lógica de até 68.416 elementos lógicos (LEs) e um máximo de portas utilizáveis de 622 como é mostrado na Tabela 1, são soluções ideais para uma ampla gama de automóveis, processamento de vídeo, comunicações, teste e medições entre outras áreas.

Table 1–1. Cyclone II FPGA Family Features

Feature	EP2C5	EP2C8	EP2C15	EP2C20	EP2C35	EP2C50	EP2C70
LEs	4,608	8,256	14,448	18,752	33,216	50,528	68,416
M4K RAM blocks (4 Kbits plus 512 parity bits	26	36	52	52	105	129	250
Total RAM bits	119,808	165,888	239,616	239,616	483,840	594,432	1,152,000
Embedded multipliers *(3)*	13	18	26	26	35	86	150
PLLs	2	2	4	4	4	4	4
Maximum user I/O pins	158	182	315	315	475	450	622

Figura 24: Características Dispositivos Família Ciclone II (**??**)

4.5.1 Arquitetura

A Figura 1 mostra a arquitetura dos dispositivos da família Cyclone II por uma visão bem alto nível. Nesta Figura é possível observar os componentes básicos da arquitetura, entre eles estão: pinos de entrada e saída, matriz lógica, blocos de memória, multiplicadores e os blocos PLL que podem multiplicar ou dividir um clock. Ainda existe uma matriz de interconexões que interliga todas as partes da arquitetura.

Os dispositivos Cyclone® II possuem um arranjo bidirecional baseado em linhas e colunas capazes de fornecer interconexões eficientes entre LABs (*Logic*

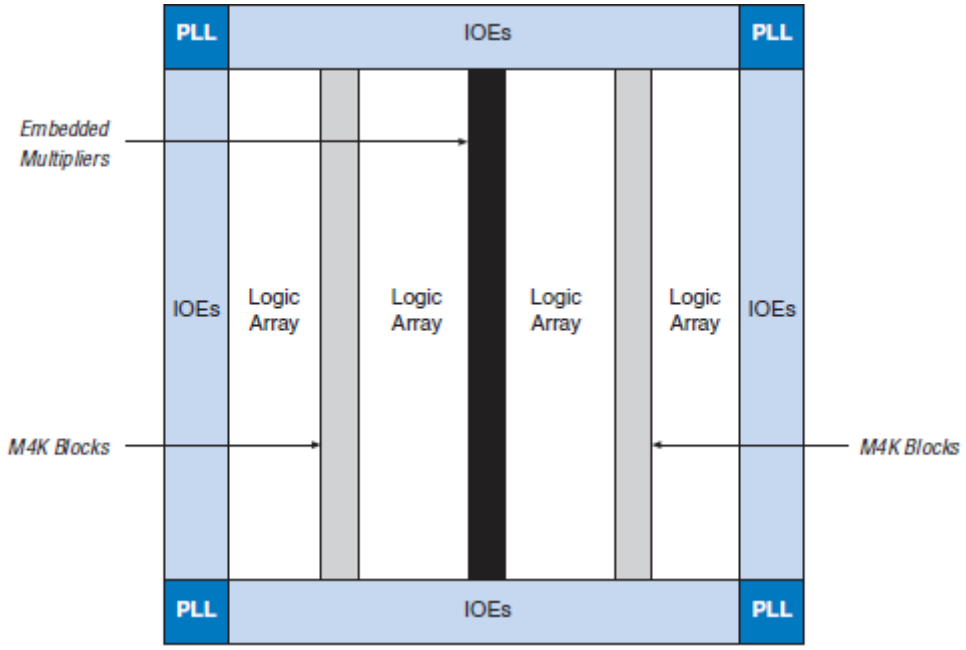

Figura 25: Arquitetura Ciclone II

Array Blocks), blocos de memória e multiplicadores (**??**).

O vetor lógico é composto por LABs, com 16 LEs (*Logic Element*) em cada LAB que são agrupados em linhas e colunas pelo dispositivo. O LE, mostrado na Figura **??**, é uma pequena unidade de execução eficiente de lógica de funções do usuário. Dispositivos Cyclone II possuem uma densidade de 4.608 para 68.416 LEs (**??**).

Os LEs possuem dois modos de operação, modo normal e o modo aritimético. O modo normal é apropriado para funções gerais da lógica e funções de combinações (*combinational functions*), já o modo aritimético é ideal para implementar contadores, somadores, acumuladores e comparadores. É importante salientar que o Quartus II automaticamente ajusta o LE para o melhor modo de operação (**??**).

Dispositivos Cyclone II oferece uma rede global de clock com mais de 4 PLLs (*Phased Locked Loops*). Esta rede é composta por mais de 16 linhas que percorrem o dispositivo fornecendo esse sinal para todos os recursos da FPGA, como

elementos de entrada / saída (IOBs), LEs, multiplicadores e blocos de memória embutidos(**??**).

Cada pino de I/O do dispositivo Cyclone II é alimentado por um IOB localizados nas extremidades das linhas e colunas do LAB em torno da periferia do dispositivo. Os pinos de I/O suportam diversos padrões, tais como padrão PCI (*Personal Computer Interface*) e o padrão para interface de memória externa do tipo DDR (*Double Data Rating*). A Figura **??** mostra o diagrama de bloco do dispositivo EP2C20 da família Cyclone II (**??**).