Universidade Virtual Africana

INFORMÁTICA APLICADA: CSI 3300

# ARQUITETURA E ORGANIZAÇÃO DE AVANÇADA DE COMPUTADORES

Dr Celestino Lopes De Barros

# Prefácio

A Universidade Virtual Africana (AVU) orgulha-se de participar do aumento do acesso à educação nos países africanos através da produção de materiais de aprendizagem de qualidade. Também estamos orgulhosos de contribuir com o conhecimento global, pois nossos Recursos Educacionais Abertos são acessados principalmente de fora do continente africano.

Este módulo foi desenvolvido como parte de um diploma e programa de graduação em Ciências da Computação Aplicada, em colaboração com 18 instituições parceiras africanas de 16 países. Um total de 156 módulos foram desenvolvidos ou traduzidos para garantir disponibilidade em inglês, francês e português. Esses módulos também foram disponibilizados como recursos de educação aberta (OER) em oer.avu.org.

Em nome da Universidade Virtual Africana e nosso patrono, nossas instituições parceiras, o Banco Africano de Desenvolvimento, convido você a usar este módulo em sua instituição, para sua própria educação, compartilhá-lo o mais amplamente possível e participar ativamente da AVU comunidades de prática de seu interesse. Estamos empenhados em estar na linha de frente do desenvolvimento e compartilhamento de recursos educacionais abertos.

A Universidade Virtual Africana (UVA) é uma Organização Pan-Africana Intergovernamental criada por carta com o mandato de aumentar significativamente o acesso a educação e treinamento superior de qualidade através do uso inovador de tecnologias de comunicação de informação. Uma Carta, que estabelece a UVA como Organização Intergovernamental, foi assinada até agora por dezenove (19) Governos Africanos - Quênia, Senegal, Mauritânia, Mali, Costa do Marfim, Tanzânia, Moçambique, República Democrática do Congo, Benin, Gana, República da Guiné, Burkina Faso, Níger, Sudão do Sul, Sudão, Gâmbia, Guiné-Bissau, Etiópia e Cabo Verde.

As seguintes instituições participaram do Programa de Informática Aplicada: (1) Université d'Abomey Calavi em Benin; (2) Université de Ougagadougou em Burkina Faso; (3) Université Lumière de Bujumbura no Burundi; (4) Universidade de Douala nos Camarões; (5) Universidade de Nouakchott na Mauritânia; (6) Université Gaston Berger no Senegal; (7) Universidade das Ciências, Técnicas e Tecnologias de Bamako no Mali (8) Instituto de Administração e Administração Pública do Gana; (9) Universidade de Ciência e Tecnologia Kwame Nkrumah em Gana; (10) Universidade Kenyatta no Quênia; (11) Universidade Egerton no Quênia; (12) Universidade de Addis Abeba na Etiópia (13) Universidade do Ruanda; (14) Universidade de Dar es Salaam na Tanzânia; (15) Universite Abdou Moumouni de Niamey no Níger; (16) Université Cheikh Anta Diop no Senegal; (17) Universidade Pedagógica em Moçambique; e (18) A Universidade da Gâmbia na Gâmbia.

Bakary Diallo

O Reitor

Universidade Virtual Africana

# Créditos de Produção

## Autor

Dr Celestino Lopes de Barros

## Par revisor(a)

Flavio Semedo

## UVA - Coordenação Académica

Dr. Marilena Cabral

## Coordenador Geral Programa de Informática Aplicada

Prof Tim  Mwololo Waema

## Coordenador do módulo

Robert Oboko

## Designers Instrucionais

Elizabeth Mbasu

Benta Ochola

Diana Tuel

## Equipa Multimédia

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

Kefa Murimi

Gerisson Mulongo

# Direitos de Autor



# Apoiado por

Projeto Multinacional II da UVA financiado pelo Banco Africano de Desenvolvimento.

# Tabela de conteúdo

# Descrição Geral do Curso

## Bem-vindo(a) a Arquitetura e Organização Avançada de Computadores

Este módulo visa apresentar e discutir alguns conceitos e princípios avançados sobre a organização interna de um computador, Pipelining; Paralelismo a nível de instrução; Multiprocessadores e Multi-cores; Paralelismo a nível de thread e Hardware reconfigurável; O foco desta disciplina está nos conceitos avançados da organização e arquitetura de Computadores.

Para melhor compreensão, este módulo está organizada em cinco unidades:

Unidade 0: Introdução à Arquitetura e Organização Avançada de Computadores, cujo propósito é verificar a compreensão e conhecimentos dos(as) estudantes sobre tópicos relacionados com este módulo e rever conceitos importantes e basilares para este módulo.

**Unidade 1**: Organização Funcional do computador é abordada e os conceitos relacionados com a organização funcional dos computadores são apresentados. Nela, são fornecidas algumas técnicas e conceitos básicos que nos irão ajudar na compreensão e análise da interação entre hardware e software.

**Unidade 2**: Multi-processamento é estudado, as noções do multiprocessamento são discutidas por forma a permitir o entendimento de como um processador pode suportar a execução simultânea de programas.

**Unidade 3:** Programação em Baixo Nível, nos permitirá identificar as limitações das arquiteturas de baixo nivel, conhecer a estrutura de programas de baixo nível e aprender a arquitetura de suporte para as linguagens de baixo, alto nível e linguagem assembler.

**Unidade 4:** Interface Entrada/Saida: permite-nos examinar o sub-sistema de entrada e saída (e/s).

## Pré-requisitos

Os pré-requisitos para este módulo são o estudo dos módulos Fundamentos de Arquitetura e Organização de Computadores e princípios de programação.

## Materiais

Os materiais necessários para completar este módulo incluem:

- Computador com conexão à internet;
- Livros listados em bibliografia;

- Links listados em bibliografia;
- Objetivos do módulo

Ao final do módulo o(a) estudante estará capacitado para:

- Definir e descrever diferentes arquiteturas de computadores;
- Descrever o funcionamento dos diferentes subsistemas de hardware;
- Identificar a informação adquirida (juntamente com os manuais) para instalação, reparação ou criação de diferentes interfaces para arquitetura de computadores.
- Escolher e comparar diferentes sistemas informáticos de alto desempenho.

## Unidades

Unidade 0: Introdução a Arquitetura e Organização Avançada de Computadores

O propósito desta unidade é verificar a compreensão dos conhecimentos dos(as) estudantes sobre tópicos relacionados com Arquitetura e Organização Avançada de Computadores.

### Unidade 1: Organização Funcional do computador

Nesta unidade serão abordados conceitos relacionados com a organização funcional do computador fornecendo técnicas e conceitos básicos que nos irão ajudar na compreensão e análise da interação entre hardware e software.

### Unidade 2: Multi-processamento

Nesta Unidade, vamos estudar as noções do multiprocessamento e entender como um processador pode suportar a execução simultânea de programas.

### Unidade 3: Programação em Baixo Nível

A Unidade 3, permite-nos identificar as limitações e estruturas das arquiteturas de baixo nivél e aprender a arquitetura de suporte das linguagens de baixo e alto nível e linguagem assembler.

### Unidade 4: Interface Entrada/Saida

Na Unidade 4 examinamos o sub-sistema de entrada e saída (E/S). Neste sub-sistema estão incluídas as interfaces de E/S, através das quais os dispositivos periféricos são conectados ao

sistema.

**Avaliação**

Em cada unidade encontram-se incluídos instrumentos de avaliação formativa a fim de verificar o progresso dos (as) estudantes.

No final de cada módulo são apresentados instrumentos de avaliação sumativa, tais como testes e trabalhos finais, que compreendem os conhecimentos construídos e as competências desenvolvidas ao estudar este módulo.

A implementação dos instrumentos de avaliação sumativa fica ao critério da instituição que oferece o curso. A estratégia de avaliação sugerida é a seguinte:

| | | |
|---|---|---|
| 1 | Cinco avaliações sumativas. Uma para cada unidade com complexidade variada e peso de 5% cada. | 25% |
| 2 | Exame Intercalar | 20% |
| 3 | Exame Final | 55% |

**Calendarização**

| Unidades | Temas e Atividades | Estimativa do tempo |
|---|---|---|
| Unidade 0: Introdução | Arquitetura de computador<br>Organização dos componentes do computador | 10 Horas |
| Unidade 1: Organização Funcional | Revisão da linguagem para descrever a transferência de registro de operações internas em computador;<br>Micro-arquiteturas - Conexões feitos por wired e por microprogramação;<br>Instruções de paralelismo em nível de instrução (ILP);<br>Desempenho do Processador e sistema de latência de memória, desempenho e eficiência. | 25 Horas |

| | | |
|---|---|---|
| Unidade 2: Multiprocessamento | Lei de Amedhl;<br>Multi-core e multi-processador;<br>Taxonomia Flynn: Estruturas e arquiteturas de multiprocessadores;<br>Sistemas de agendamento multiprocessador; | 25 Horas |
| Unidade 3: Programação em Baixo Nível | Estrutura de programas de baixo nível;<br>Vantagens e Limitações de arquitecturas de baixo nível;<br>Arquitetura de suporte de linguagens de baixo nível e de alto nível. | 27 Horas |
| Unidade 4: Interface Entrada/Saida | Interação entre Processador e Interfaces de E/S;<br>Organização de uma Interface de E/S<br>Técnicas de Transferência de Dados<br>Padrões de Barramentos | 30 Horas |
| Exame Final | Avaliação final | 3 Horas |
| TOTAL | | 120 Horas |

**Leituras e outros Recursos**

As leituras e outros recursos deste curso são:

## Na Unidade 0, Introdução a Arquitetura e Organização Avançada de Computadores

Leituras e outros recursos obrigatórios:

- TANENBAUM, Andrew S. Organização estruturada de computadores. São Paulo: Pearson Prentice Hall, 2007, Cap 2.
- MONTEIRO, Mário A. Introdução à organização de computadores. Rio de Janeiro: LTC, 2007, Cap. 1, Cap. 2 e Cap 10.

## Na Unidade 1, Organização Funcional

Leituras e outros recursos obrigatórios:

- Huang, Ing-Jer. Despain, Alvin M. "Hardware/Software Resolution of Pipeline Hazards in Pipeline Synthesis of Instruction Set Processors", IEEE, 1993.
- TANENBAUM, Andrew S. Organização estruturada de computadores. São Paulo: Pearson Prentice Hall, 2007, Cap 2.
- MONTEIRO, Mário A. Introdução à organização de computadores. Rio de Janeiro: LTC, 2007, Cap. 1, Cap. 2 e Cap 10.
- MURDOCCA, Miles J. Introdução à arquitetura de computadores. Rio de Janeiro: Campus, 2000 Cap. 8.
- PATTERSON, David A. HENNESSY, John L. Organização e projeto de computadores: a interface hardware/software. 2. ed. Rio de Janeiro: LTC, 2000. 551p.
- STALLINGS, W. Arquitetura e organização de computadores. São Paulo: Prentice Hall, 2002.

## Unidade 2, Multi-processamento

Leituras e outros recursos obrigatórios:

- Alves, Marco, Avaliação do Compartilhamento das Memórias Cache no Desempenho de Arquiteturas Multi-Core –
- http://www.lume.ufrgs.br/bitstream/handle/10183/16129/000697073.pdf?sequence=1 consultado em 4 de Janeiro de 2015
- Breshears, C. The Art of Concurrency. San Francisco, CA, USA: O'Reilly, 2009.
- Patterson, D. A.; hennessy, J. L. Computer Architecture: a quantitative approach. Fourth edition. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.2007.

## Unidade 3, Programação em Baixo Nível

Leituras e outros recursos obrigatórios:

- PATTERSON, David A.; HENNESSY, John L. Organização e projeto de computadores: a interface hardware/software. 2. ed. Rio de Janeiro: LTC, 2000. Cap 5.
- TANENBAUM, Andrew S. Organização estruturada de computadores. São Paulo: Pearson Prentice Hall, 2007, Cap 3.
- http://pt.wikipedia.org/wiki/ acedido em 27-02-2016

## Unidade 4, Interface Entrada/Saída

Leituras e outros recursos obrigatórios:

- TANENBAUM, Andrew S. Organização estruturada de computadores. São Paulo: Pearson Prentice Hall, 2007, Cap 4.
- http://pt.wikipedia.org/wiki/ acedido em 27-02-2016

# Unidade 0. Introdução à Arquitetura e Organização Avançada de Computadores

## Introdução à Unidade

O propósito desta unidade é verificar o seu nível de conhecimentos e grau de compreensão dos tópicos relacionados com este módulo.

As expectativas são de que você desenvolva uma base de conhecimentos prévios sobre os Fundamentos de Arquitetura de Computadores e Principios de Programação.

## Objetivos da Unidade

Após a conclusão desta unidade, você deverá ser capaz de:

- Identificar o modo e os princípios básicos de funcionamento de sistemas informáticos;
- Analisar o desempenho do sistema informático;
- Identificar os conceito por detrás das técnicas avançcadas de pipelining;
- Identificar o estado da arte do desenho do sistema de memória;
- Analisar a forma e os princípios pelos quais os dispositivos E/S são acedidos.

**TERMOS-CHAVE**

**I/O Entrada/Saída:** é um termo utilizado em informática para indicar a entrada (inserção) de dados por meio de algum código ou programa, para algum outro programa ou hardware, bem como a sua saída (obtenção de dados) ou devolução de dados, como resultado de alguma operação de algum programa, resultado de alguma entrada.

**Memória:** são todos os dispositivos que permitem um computador guardar dados, temporária ou permanentemente.

### Avaliação da Unidade

As seguintes questões nos ajudarão a avaliar os seus atuais conhecimentos sobre Fundamentos de Arquitetura e Organização de Computadores que são pré-requisitos do módulo Arquitetura e Organização Avançada de Computadores.

Defina por suas palavras os seguintes termos:

- Arquitetura de Computadores.
- Organização de computadores.
- Dispositivos Perifericos.
- Descreva as camadas no projeto de arquitetura de computadores.

**Resposta à questão N° 1.a)**

Arquitetura de Computadores é o projeto conceitual fundamental da estrutura operacional de um sistema informático. É o estudo dos requisitos necessários para que um computador funcione na perfeição e de como organizar os diferentes componentes para obter melhores desempenhos.

**Resposta à questão N° 1.b)**

O termo organização de computadores refere-se às unidades funcionais e às suas interconexões que compoêm o computador. Desta forma, uma mesma arquitetura pode ser implementada através de diferentes organizações. A arquitetura de um computador estabelece o modelo da organização e funcionamento de um sistema de processamento, com todas suas partes divididas em seções, interagindo entre si.

**Resposta à questão N° 1.c)**

Periféricos são dispositivos instalados no computador, cuja função é auxiliar a interação homem/máquina. Estes dispositivos poderão estar ligados ao computador ou mesmo dentro do próprio gabinete. O gabinete é uma caixa metálica horizontal ou vertical, tem a função de servir como suporte à placa-mãe, leitores, discos e outros dispositivos eletrônicos. Nele são conectados os periféricos.

**Resposta à questão N° 2)**

Os conceitos de camadas no projeto de arquitetura são descritos como a seguir:

Problema complexo pode ser segmentado em tamanho e complexidade menores, possibilitando assim a melhor gestão;

Cada camada é especializada na realização das suas tarefas específicas.

Camadas superiores podem partilhar serviços de uma camada inferior. Essa partilha permite reutilizar funcionalidades.

A segmentação lógica possibilita o desenvolvimento em equipa. Uma equipa de programadores pode desenvolver um sistema. Esta poderá ser sub-dividida sendo que cada sub-grupo poderá desempenhar as suas funções específicas desde que sejam definidos, claramente, os limites.

# Unidade 1. Organização funcional do Computador

## Introdução à Unidade

Nesta unidade trataremos de conceitos relacionados com a organização funcional do computador fornecendo técnicas e conceitos básicos que nos irão ajudar na compreensão e análise da interação entre hardware e software. Daremos um enfoque especial à arquitetura do processador que se encontra organizado em duas unidades: a seção de processamento e a seção de controlo.

Esta unidade descreve os principais componentes em cada uma destas unidades. Dentro deste contexto, também se relata como um processador executa as instruções de um programa.

## Objetivos da Unidade

Após a conclusão desta unidade, você deverá ser capaz de:

Compreender e descrever a organização funcional do computador;

Reconhecer os princípios básicos de organização, funcionamento e desempenho de sistemas informáticos modernos;

Compreender e delinear o projeto da arquitetura de computador.

**TERMOS-CHAVE**

**CPU:** acrónimo de Central Processing Unit - Também conhecido como processador, é a parte do computador, que executa as instruções de um programa.

**Memória Cache:** É uma pequena memória localizada junto ao processador. Surgiu quando a memória de acesso aleatório não estava a acompanhar o desenvolvimento rápido do processador. Auxilia na execução das instruções/processos.

**Operações Internas**: Conjuto de instruções que são executadas na parte interna do computador.

**Microarquitetura:** É a forma como um determinado conjunto de instruções (ISA) é implementado em um processador, podendo ser implementado com microarquitecturas diferentes. As implementações podem variar devido a diferentes objetivos de um dado projeto ou a mudanças na tecnologia.

**MIPS:** acrónimo de Microprocessor without interlocked pipeline stages é uma arquitetura de microprocessadores RISC desenvolvida pela MIPS Computer Systems.

**RAM:** Random Access Memory - é um tipo de memória que permite a leitura e a escrita, É utilizada como memória primária em sistemas eletrônicos digitais.

**ROM:** Read Only Memory - é um tipo de memória que permite apenas a leitura, ou seja, as suas informações são gravadas pelo fabricante uma única vez e após isso não podem ser alteradas ou apagadas, somente lidas.

**Sistema operativo** - é um programa ou um conjunto de programas cuja função é gerir os recursos do sistema, fornecendo uma interface entre o computador e o utilizador.

**Performance de Sistema:** É a verificação do desempenho do sistema.

**Atividades de Aprendizagem**

## Atividade 1 – Processador: seção de Processamento e de controlo

**Introdução**

O processador é o componente vital do sistema informático, é responsável pela realização das operações de processamento e de controlo, durante a execução de um aplicativo.

Um programa, para ser efetivamente executado pelo processador, deve ser constituído por um conjunto de instruções de máquina. Para que a execução tenha início, as instruções devem estar armazenadas na memória.

Nesta atividade, vamos discutir a seção do processamento e controlo do processador, formas de implementar a unidade de controlo, a linguagem de descrição e transferência de registo e operações internas no computador.

**Detalhes da atividade**

O processador executa muitas tarefas, de entre outras, destacamos: a procura na memória das instruções a serem executadas; interpretação da operação que a instrução está a explicitar; procura dos dados e onde estão armazenados; execução efetiva da operação com os dados e o resultado do armazenamento no local definido pela instrução; reiniciar o processo, procurar a próxima instrução.

O processador é composto por algumas seções e nesta atividade vamos destacar a seção de processamento e a de controlo.

**A Seção de Processamento**

A seção de processamento é formada basicamente pela Unidade Lógica e Aritmética (ALU) e por diversos registradores. Estes componentes normalmente estão organizados conforme mostra a Figura 1.1.

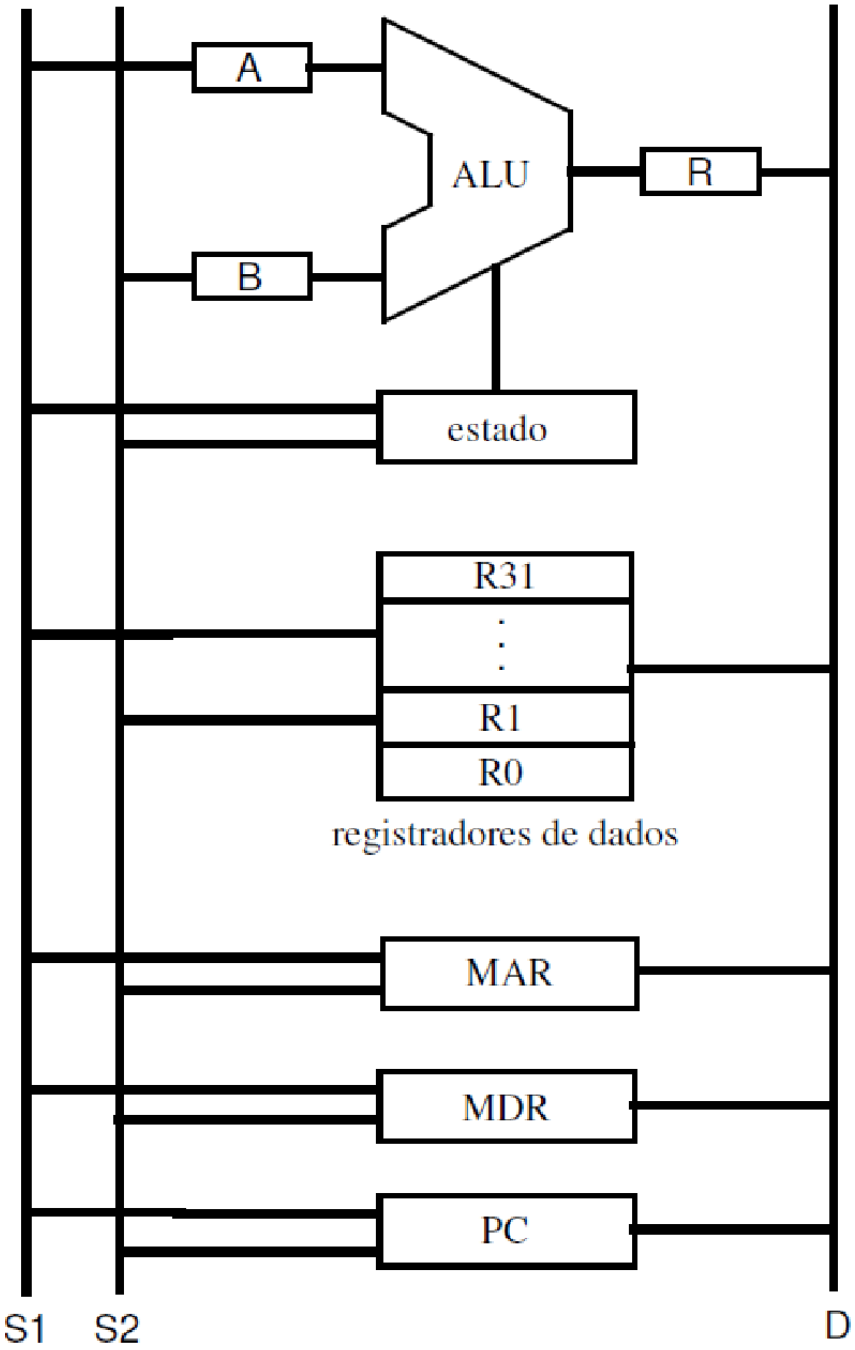



Figura 1.1: Componentes da seção de processamento

**Fonte: Adaptado de TANENBAUM, 2007**

A ALU realiza as operações aritméticas, tais como adição e subtração, e operações lógicas, tais como and, or, not. Podemos dizer que a ALU é o componente da arquitetura que, de fato, processa os dados. Os registradores são utilizados para armazenar informações internamente no processador. Um registrador pode ser utilizado tanto para acesso de leitura quanto para acesso de escrita: uma informação é armazenada no registrador em uma operação de escrita, enquanto a informação contida no registrador é recuperada em uma operação de leitura.

### A Execução de Instruções

Tendo examinado os componentes e a organização da seção de processamento, podemos agora analisar como as instruções são executadas. A execução de uma instrução envolve a realização de uma sequência de passos, que podemos chamar de passos de execução. Em geral, a execução de uma instrução envolve quatro passos, como mostra a Figura 1.2.

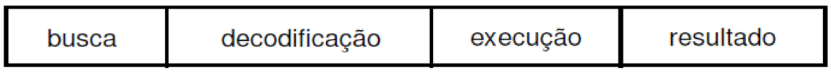



Figura 1.2: Passos na execução de uma instrução

**Fonte: Adaptado de TANENBAUM, 2007**

No primeiro passo, denominado busca, o processador realiza o acesso ao código binário da instrução, armazenado na memória principal. A etapa seguinte é a decodificação da instrução, na qual as informações contidas no código da instrução são interpretadas. Em algumas arquiteturas, neste passo também são acessados os dados utilizados pela instrução. Após a decodificação, a execução da instrução entra no terceiro passo, denominado execução, no qual a operação indicada pela instrução (por exemplo, uma operação na ALU) é efetuada. Finalmente no quarto passo, chamado resultado, é armazenado, em um registrador ou na memória, o resultado produzido pela instrução.

Cada passo de execução envolve a realização de várias operações básicas.

As operações básicas acontecem dentro da seção de processamento, sob a coordenação da seção de controlo. Existem quatro principais tipos de operações básicas:

- Transferência de dados entre os registradores e a ALU;
- Transferência de dados entre os registradores;
- Transferência de dados entre os registradores e a memória;

**Operações aritméticas e lógicas realizadas pela ALU.**

Para tornar mais claro o mecanismo de execução de instruções, considere uma arquitetura com uma seção de processamento idêntica à da Figura 1.1. Considere também que a arquitetura oferece quatro tipos de instruções: instruções aritméticas e lógicas, instruções de desvio incondicional e condicional, e instruções de acesso à memória. Exemplos destes tipos de instruções aparecem no quadro da Figura 1.3.

| Tipo | Exemplo | Descrição |
|---|---|---|
| aritmética/lógica | `ADD Rs1,Rs2,Rd` | O conteúdo dos registradores Rs1 e Rs2devem ser somados e o resultado armazenado em Rd. |
| desvio incondicional | `JMP dst` | A próxima instrução a ser executada deve ser a que se encontra na locação de memória com endereço dst. |
| desvio condicional | `JZ dst` | A próxima instrução a ser executada deve ser a que se encontra no endereço dst, caso o bit Z no registrador de estado esteja ativado. |
| acesso à memória | `LOAD end,R1` | O dado armazenado na locação de memória com endereço end deve ser transferido para o registrador R1. |

Figura 1.3: Exemplos de instruções

**Fonte: Adaptado de (TANENBAUM, 2007)**

A Figura 1.4 apresenta um exemplo hipotético relacionando, para cada tipo de instrução, as operações básicas que acontecem dentro de cada passo de execução. Nesta figura, Ry ← Rx representa a transferência do conteúdo do registrador Rx para o registrador Ry. M[R] denota o conteúdo da locação de memória cujo endereço está no registrador R. Finalmente, IR (instruction register) representa um registrador especial que recebe o código da instrução a ser executada, enquanto PC é o contador de programa.

| | Aritméticas e Lógicas | Desvios Incondicionais | Desvios Condicionais | Acessos à Memória |
|---|---|---|---|---|
| Busca | MAR ← PC<br>MDR ← M[MAR]<br>IR ← MDR<br>PC++ | MAR ← PC<br>MDR ← M[MAR]<br>IR ← MDR<br>PC++ | MAR ← PC<br>MDR ← M[MAR]<br>IR ← MDR<br>PC++ | MAR ← PC<br>MDR ← M[MAR]<br>IR ← MDR<br>PC++ |
| Decodificação | decod<br>A ← Rs1<br>B ← Rs2 | decod | decod | decod |
| Execução | R ← A op B | PC ← destino | cond<br>se (cond) PC ← destino | MAR ← end<br>MDR ← Rs (E)<br>M[MAR] ← MDR (E)<br>MDR ← M[MAR] (L) |
| Resultado | Rd ← R | | | Rd ← MDR (L) |

Figura 1.4: Operações básicas na execução de instruções

**Fonte: Adaptado de TANENBAUM, 2007**

O passo de procura é idêntico para todos os tipos de instruções e envolve quatro operações básicas: (1) o conteúdo do contador de programa é transferido para o registrador de endereço de memória MAR; (2) é realizado o acesso à memória, utilizando o endereço em MAR; (3) o código de instrução recebido da memória, temporariamente armazenado em MDR, é transferido para o registrador de instrução IR (este registrador faz parte da seção de controle; (4) contador de programa é incrementado, passando a indicar a próxima instrução onde será feito e executado o acesso.

Concluído o passo de busca, inicia-se o passo de decodificação da instrução armazenada no IR. A interpretação do código da instrução é indicada por "decod" na Figura 1.4. Como mencionado, em algumas arquiteturas este passo também inclui o acesso aos operandos da instrução. Na Figura 1.4, isto é indicado pela transferência do conteúdo dos registradores de dados Rs1 e Rs2 para os registradores temporários A e B, respectivamente, no caso de instruções aritméticas e lógicas.

O próximo passo é o de operação. As operações básicas neste passo dependem inteiramente do tipo de instrução que está a ser executada. No caso das instruções aritméticas e lógicas, este passo corresponde à execução pela ALU da operação indicada na instrução, utilizando como operandos o conteúdo dos registradores A e B, com armazenamento do resultado no registrador R.

Em instruções de desvio incondicional, apenas uma operação básica é realizada: o endereço destino é carregado no contador de programa. Como o contador de programa indica a próxima instrução a ser executada, isto resulta em um desvio para a instrução armazenada no endereço destino. Em desvios condicionais, o contador de programa é modificado somente se a condição de desvio for verdadeira. A avaliação da condição de desvio é indicada por "cond", na Figura 1.4. O endereço destino é carregado no contador de programa apenas se a condição de desvio testada for verdadeira.

Em instruções de acesso à memória, o passo de execução inicia-se com a transferência do endereço da locação onde será feito o acesso para o registrador MAR. As demais operações básicas dependem se o acesso é de escrita (indicadas por E) ou de leitura (indicadas por L). No caso de uma escrita, são executadas duas operações básicas: a transferência do dado a ser escrito para o registrador MDR e a escrita na memória propriamente dita. No caso de uma leitura, é realizado o acesso à locação de memória e o dado obtido é armazenado no registrador MDR.

O último passo é o de armazenamento do resultado. Em instruções aritméticas e lógicas, o resultado no registrador R é transferido para o registrador destino, indicado na instrução. Em instruções de leitura à memória, o dado que se encontra no registrador MDR é transferido para o registrador destino.

É importante salientar que o quadro na Figura 1.4 é extremamente simplificado. Por exemplo, na prática um acesso à memória não é feito por uma única operação básica como indicado, mas normalmente requer várias operações básicas. No entanto, este quadro reflete corretamente como a execução de uma instrução é logicamente organizada.

### A Seção de Controle

Como mencionado anteriormente, as operações básicas que ocorrem dentro da seção de processamento são todas comandadas pela seção de controlo. Ao efetuar a pesquisa da instrução, a unidade de controlo interpreta a instrução de modo a identificar quais as operações básicas que devem ser realizadas, e ativa sinais de controlo que fazem uma operação básica de fato acontecer.

A Figura 1.5 apresenta um diagrama em blocos da seção de controlo e, de forma bastante simplificada e ilustrativa, a sua interligação com a seção de processamento. Como mostra a figura, a seção de controlo é formada basicamente pela unidade de controlo e pelo registrador de instrução, ou IR (Instruction Register). A interpretação do código da instrução e a ativação dos sinais de controlo são realizados pela unidade de controlo. Os sinais de controlo que são ativados, bem como a sequência com que são ativados, dependem de cada instrução em particular.

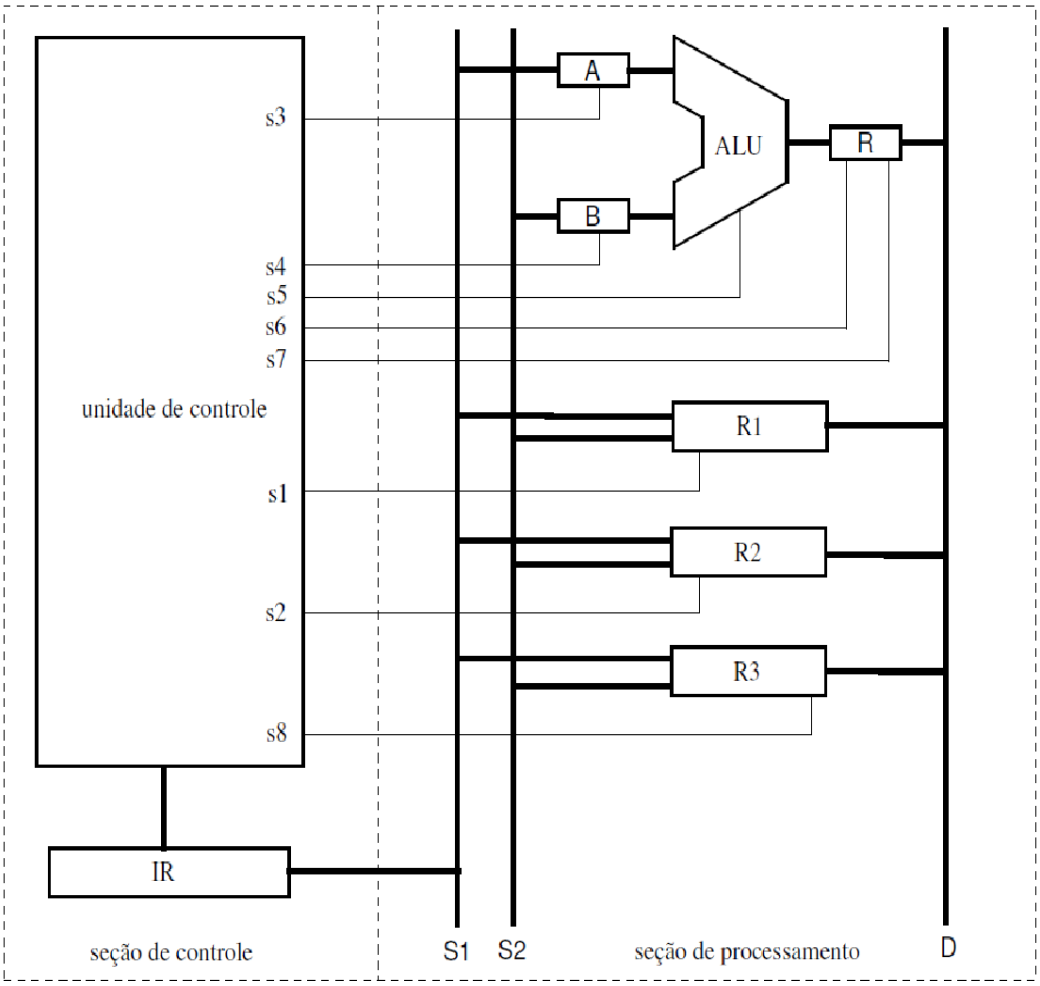



Figura 1.5: A seção de controlo e a parte de processamento

**Fonte: Adaptado de TANENBAUM, 2007**

A título de exemplo, vamos verificar como a execução da instrução ADD R1,R2,R3 é direcionada pela unidade de controlo. Para tanto, a seção de processamento na figura 1.5 foi representada com apenas três registradores de dados envolvidos na execução desta instrução. A execução desta instrução requer as seguintes operações básicas:

Transferência do conteúdo do registrador de dados R1 para o registrador temporário A;

Transferência do conteúdo do registrador de dados R2 para o registrador temporário B;

Adição dos dados armazenados nos registradores A e B e armazenamento do resultado no registrador R;

Transferência do conteúdo do registrador R para o registrador R3.

A sequência de ativação dos sinais de controlo, com as operações básicas correspondentes, é apresentada na Figura 1.6.

| operação básica | sinal de controle | descrição da operação básica |
|---|---|---|
| (1) (2) | s1, s2 | coloca o conteúdo de R1, R2 para os barramentos S1,S2, respectivamente. |
| | s3,s4 | armazena a informação presente nos barramentos S1, S2 em A,B, respectivamente. |
| (3) | s5 | seleciona a operação de soma na ALU. |
| | s6 | armazena o resultado produzido pela ALU em R. |
| (4) | s7 | coloca o conteúdo de R para o barramento D. |
| | s8 | armazena a informação presente no barramento D em R3. |

Figura 1.6: Ativação dos sinais de controlo e operações básicas correspondentes

**Fonte: Adaptado de TANENBAUM, 2007**

Este é um exemplo típico de como a Unidade de Controlo coordena a execução das operações básicas na seção de processamento. Para cada registrador existem sinais que controlam a leitura e a escrita do registrador. Outros sinais indicam à ALU a operação aritmética ou lógica que deve realizada. Para qualquer outro componente na seção de processamento existem os sinais de controlo necessários. Para executar uma operação básica, a unidade de controlo simplesmente ativa os sinais apropriados na sequência correta.

Após a leitura dos detalhes da atividade e para auxiliar a compreensão integral dos conteúdos, recomendamos que assista aos videos:

- "Hardware para iniciantes – Processado". Disponível em: https://www.youtube.com/watch?v=DFMCcuhWiXM consultado em 26-02-2015.
- "Como funciona a arquitetura de um processador Intel Pentium". Disponível em: https://www.youtube.com/watch?v=LN6LuhRYzuA consultado em 16-02-2015.
- Faça um pequeno comentário escrito sobre o vídeo assistido e ao texto lido e envie-o em forma de apresentação digital ao (à) instrutor (a) através do correio eletrónico.

## Conclusão

Nesta atividade demos um enfoque especial à arquitetura do processador que se encontra organizado em duas seções: A seção de processamento e a de controlo.

A seção de processamento é formada basicamente pela unidade lógica e aritmética (ALU) e por diversos registradores. A ALU é o componente da arquitetura que, de fato, processa os dados. E os registradores são utilizados para armazenar informações internamente no processador.

As operações básicas que ocorrem dentro da seção de processamento são todas comandadas pela seção de controlo.

**Avaliação da Actividade**

Este conteúdo será avaliado na avaliação sumativa da Unidade 1 que tem o peso de 5%.

## Atividade 2 – Formas de implementar a Unidade de Controlo

**Introdução**

Segundo o site da Universidade Federal do Pará, acedido em Fevereiro de 2017, Unidade de Controlo é a unidade que armazena a posição de memória que contém a instrução que o computador está executando nesse momento. Ela informa à ULA qual operação a executar, buscando a informação (da memória) que a ULA precisa para executá-la.

Depois, transfere o resultado de volta para o local apropriado da memória. Temos duas formas de o implementar: utizando lógica aleatória ou hardwired ou utizando microprogramação.Nesta atividade vamos estudar as formas de implementar a Unidade de Controlo.

**Detalhes da atividade**

Existem basicamente duas maneiras de implementar uma unidade de controlo:

- A primeira delas é utizando lógica aleatória (hardwired control).
- E a outra é a utilização de microprogramação.

A Figura 1.7 ilustra a estrutura típica de uma Unidade de Controlo implementada com lógica aleatória.

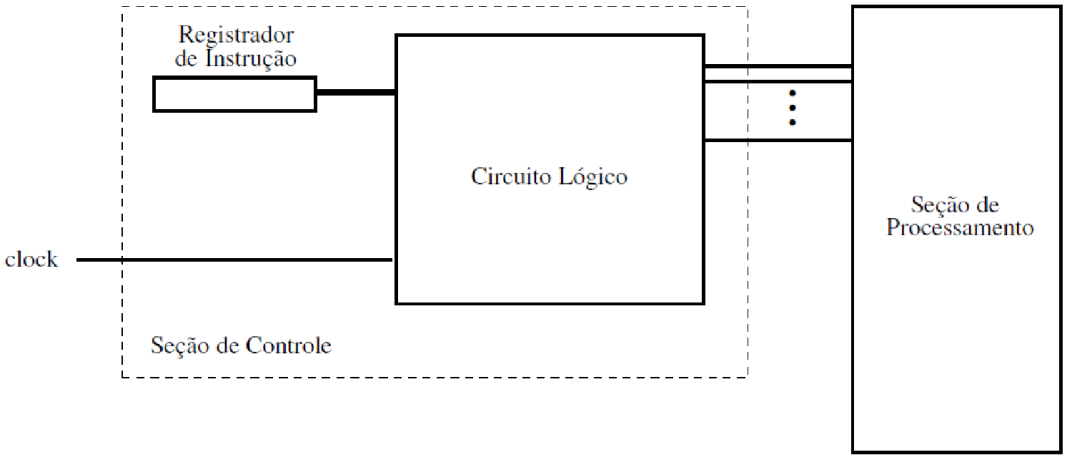



Figura 1.7: Organização de uma unidade de controlo implementada com lógica aleatória.

**Fonte: Adaptado de TANENBAUM, 2007**

Na implementação por lógica aleatória, a Unidade de Controlo é formada por um único circuito lógico, cuja entrada é o código de instrução armazenado em IR e cujas saídas são os próprios sinais de controlo que comandam as operações básicas na seção de processamento. De acordo com o código da instrução, a cada ciclo de relógio este circuito ativa os sinais de controlo que comandam as operações básicas que devem ser realizadas naquele ciclo.

O inconveniente desta forma de implementação é que a complexidade do circuito de controlo, em termos do número de dispositivos lógicos, aumenta rapidamente com o número de instruções oferecidas pela arquitetura e com o número de operações básicas que devem ser realizadas na execução de uma instrução.

Em arquiteturas com instruções funcionalmente complexas, o projeto de uma unidade de controlo com lógica aleatória torna-se muito difícil e propenso a erros.

A técnica de microprogramação corrige esta desvantagem da implementação com lógica aleatória. Nela, cada instrução oferecida pela arquitetura, que passa a ser chamada de macroinstrução, é na realidade executada por uma sequência de instruções primitivas, extremamente simples, chamadas microinstruções. Os próprios bits das microinstruções são utilizados para ativar e desativar os sinais de controlo que comandam as operações básicas. A sequência de microinstruções que executa uma macroinstrução forma uma microrotina. Utilizando de uma interpretação mais simples, podemos considerar que a execução de uma macroinstrução consiste na chamada de uma microrotina, feita pela unidade de controlo. As microinstruções da microrotina executam as operações básicas associadas à macroinstrução.

A Figura 1.8 mostra a estrutura de uma unidade de controlo implementada com a técnica de microprogramação.

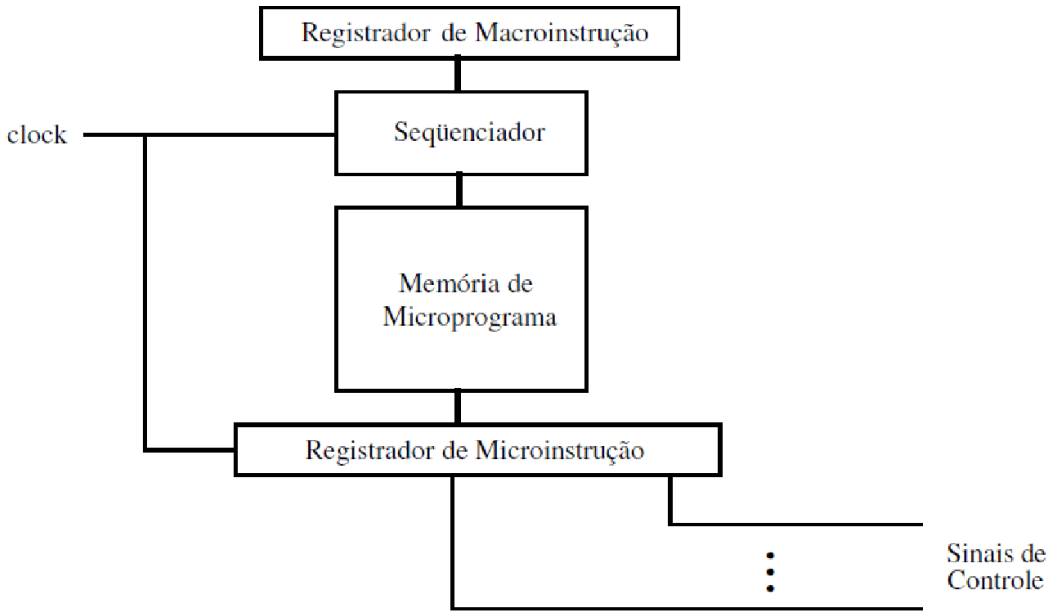



Figura 1.8: Organização de uma unidade de controlo microprogramada.

**Fonte: Adaptado de TANENBAUM, 2007**

As microrotinas encontram-se armazenadas na memória do microprograma.

Quando o código da macroinstrução é armazenado no registrador de (macro) instrução, o sequenciador interpreta este código e determina o endereço de entrada da microrotina que executa aquela macroinstrução. O sequenciador fornece, a cada ciclo de clock, o endereço da próxima microinstrução a ser executada. Após o acesso à microinstrução, ela é armazenada no registrador de microinstrução. Alguns bits da microinstrução são utilizados diretamente para comandar os sinais de controlo para a seção de processamento.

Outros bits são utilizados pelo sequenciador para determinar a próxima microinstrução a ser executada.

A execução de uma microinstrução envolve o acesso da microinstrução na memória de microprograma, o armazenamento no registrador de microinstrução e a realização das operações básicas comandadas pelos bits da microinstrução. Uma nova microinstrução é executada a cada ciclo de clock. Quando a execução de uma microrotina é concluída, uma nova macroinstrução é acedida na memória principal e armazenada no registrador de instrução, e é iniciada a execução de uma nova microrotina.

A vantagem da microprogramação está no fato que a implementação das instruções é reduzida basicamente à escrita das microrotinas que serão gravadas na memória de microprograma. Esta vantagem se torna especialmente significativa quando a arquitetura oferece um grande número de instruções e estas instruções são complexas, ou seja, a sua execução envolve um grande número de operações básicas. Neste caso, o projeto de um complicado circuito lógico, como aconteceria na implementação com lógica aleatória, é substituído pela escrita das microrotinas, uma tarefa comparativamente bem mais simples.

Após a leitura cuidada do texto para esta atividade responda às questões abaixo:

Quais a desvantagem da implementação da unidade de controlo através da lógica aleatória?

Quais das formas de implementação de unidade controlo (lógica aleatória ou microprogramação) é a melhor? Justifique?

**Resposta a questão Nº 1**

As desvantagens da implementação de unidade de controlo através da lógica aleatória são que a complexidade do circuito de controlo, em termos do número de dispositivos lógicos, aumenta rapidamente com o número de instruções oferecidas pela arquitetura e com o número de operações básicas que devem ser realizadas na execução de uma instrução. Outra desvantagem é que, em arquiteturas com instruções funcionalmente complexas, o projeto de uma unidade de controlo com lógica aleatória torna-se muito difícil e propenso a erros.

**Resposta a questão Nº 2**

A melhor forma de implementar a unidade de controlo é recorrendo à microprogramação. Porque a microprogramação permite implementar as instruções de forma reduzida através das microrotinas que serão gravadas na memória do microprograma.

Esta vantagem é mais visível sobretudo quando a arquitetura oferece um grande número de instruções e estas instruções são complexas, ou seja, a sua execução envolve um grande número de operações básicas. Neste caso, o projeto de um complicado circuito lógico, como aconteceria na implementação com lógica aleatória, é recomendável que seja substituído pela escrita das microrotinas que são tarefas comparativamente bem mais simples.

## Conclusão

Nesta atividade abordamos as formas de implementação da Unidade de Controlo e concluímos que temos duas formas de a implementar: utizando lógica aleatória/hardwired ou utizando microprogramação.

Concluimos ainda que a utilização de microprogramação é mais adequada para implementação da unidade de controlo porque ela permite implementar instruções de forma reduzida através das microrotinas.

**Avaliação**

Este conteúdo será avaliado na avaliação sumativa da Unidade 1 que tem o peso de 5%.

## Atividade 3 – Paralelismo em nível de instrução (ILP)

**Introdução**

A necessidade de os computadores executarem, o mais rapidamente possível, sequências de instruções, tem suscitado muitas investigações e várias abordagens. Umas atuam na implementação física do processador, tal como a utilização de transistores mais rápidos. Outras actuam realizando melhorias na arquitetura.  Uma forma de melhorar a arquitetura é adicionar paralelismo em nível de instrução. Processadores com paralelismo em nível de instrução podem executar instruções de forma paralela e, idealmente, de forma transparente para o programador.

Esta atividade tem como  foco analisar o paralelismo ao nível de instruções.

**Detalhes da atividade**

A arquitectura dos processadores modernos baseia-se fortemente na capacidade em executar várias instruções em cada ciclo de relógio, o que normalmente se designa por paralelismo ao nível da instrução. Esta designação surge porque as instruções são executadas em paralelo. Duas técnicas são frequentemente utilizadas para a execução de instruções em paralelo:

Execução encadeada de instruções – cada instrução é executada em várias fases, sendo a execução de várias instruções sobreposta, como numa linha de montagem; as várias instruções executam em paralelo, mas em fases diferentes;

Execução super-escalar de instruções – as instruções são executadas em paralelo, o que envolve a duplicação de unidades funcionais para suportar a combinação de instruções pretendida.

Estas duas técnicas são geralmente combinadas, sendo ambas utilizadas na arquitectura dos processadores modernos. A Figura 1.9 apresenta uma comparação das duas técnicas para um processador que executa as instruções em 5 fases:

**Pesquisa da instrução (IF)**

- Leitura dos registos e descodificação das instruções (ID)
- Execução da operação ou cálculo de endereço (EXE)
- Acesso ao operando em memória (MEM)
- Escrita do resultado em registo (WB)

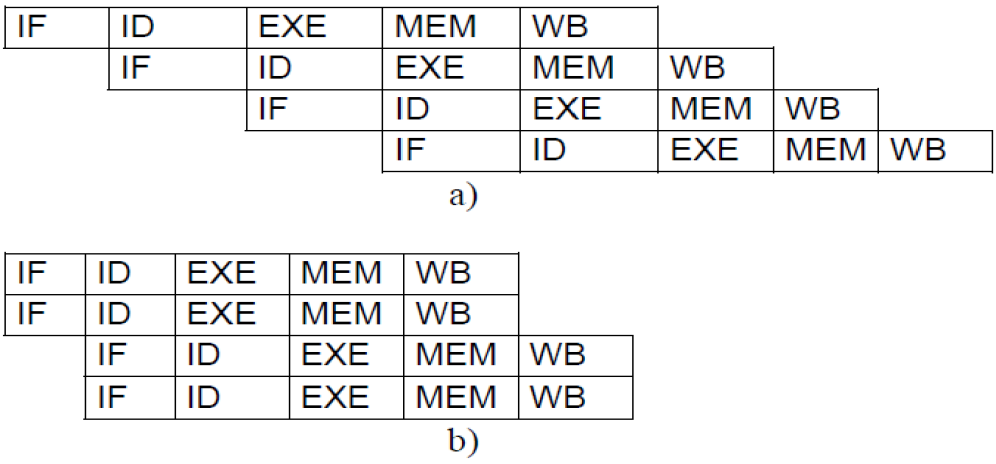



Figura 1.9: – a) Execução encadeada de instruções b) execução super-escalar

**Fonte: Adaptado pelo autor**

O tempo de execução de um programa é dado por:

- Texe = #instruções x CPI x Tcc
- A execução de instruções em cadeia permite reduzir a duração do ciclo de relógio (Tcc) do processador, ou seja, aumentar a sua frequência. Por outro lado, a execução super-escalar de instruções aumenta o número de instruções realizadas em cada ciclo (IPC) de relógio, ou seja, diminui o CPI.
- Note que qualquer uma destas duas alternativas permite reduzir o tempo de execução da aplicação.
- Uma técnica muito utilizada para alcançar o paralelismo em nível de instruções é a utilização de pipeline. A técnica do pipeline consegue ganhos porque uma instrução de máquina pode ser dividida em uma sequência de etapas intermediárias.
- Em uma arquitetura de microprocessador sem estágios intertravados de pipeline (MIPS), por exemplo, cada instrução pode ser dividida em cinco etapas: procura, decodificação, execução, escrita em memória e escrita nos registradores. Como cada instrução está a realizar apenas uma dessas etapas em um instante de tempo, mais de uma instrução pode ser executada ao mesmo tempo desde que elas não estejam na mesma etapa.

- Pipelines apresentam um problema conhecido como hazards. Estes degradam a performance por impedir que as tarefas subsequentes sejam executadas no seu tempo correto (Huang, 1993).
- Os hazards acontecem devido a interdependência entre as instruções e os dados utilizados pelas mesmas, os recursos de hardware necessários para a execução das mesmas e por fim pela dependência entre as instruções e o fluxo das instruções.
- Os structural hazards acontecem quando instruções que estão a ser executadas de forma paralela e em etapas diferentes necessitam do mesmo recurso de hardware. Por exemplo, uma instrução que está na etapa de escrita em memória está a fazer a utilização do hardware da memória, enquanto uma instrução que está na etapa de procura também está a fazer a utilização do mesmo hardware. Se estas duas instruções forem executadas em um pipeline pode acontecer de ocorrer uma tentativa de execução ao mesmo tempo, temos então um structural hazard.
- Outro tipo de hazard são os data hazard. Neste, o problema ocorre quando uma instrução necessita de dados calculados pela instrução anterior. Nestas condições pode ser que o acesso aos dados seja feito antes da instrução anterior terminar de calculá-los, caracterizando um data hazard. O último tipo são os branch hazards. Nestes, a próxima instrução a ser executada depende do resultado da instrução atual (instrução de desvio). Se o desvio ocorrer, os resultados das operações das instruções seguintes que estavam no pipeline devem ser descartados pois descobriu-se tardiamente que estas não estavam no fluxo correto de execução do programa.
- Detetar e solucionar os hazards dos pipelines é um dos maiores problemas na construção dos mesmos, onde a solução mais simples pode esvaziar o pipeline antes da execução da instruções seguinte (Huang, 1993).
- A técnica da inserção de "bolhas" no pipeline, onde a execução da instrução seguinte é adiada por algumas etapas também é constantemente utilizada.
- Após a leitura do texto da atividade 1.3 responda as seguintes questões:
- Qual a principal vantagem da utilização de paralelismo a nível de instruções.
- Uma das técnica utilizada para alcançar o paralelismo em nível de instruções é a utilização de pipeline, Quais os principais problemas apresentadas por esta técnica e porque acontecem?

**Resposta à questão N° 1**

A principal vantagem da utilização de paralelismo a nível de instruções é a necessidade dos computadores executarem o mais rapidamente possível sequências de intruções.

**Resposta à questão N° 2**

Os principais problemas apresentados por esta técnica são os hazards que acontecem devido à interdependência entre as instruções e os dados utilizados,

os recursos de hardware necessários para a execução e por fim pela dependência entre as instruções e o fluxo das instruções. Possuem três tipos: Structural hazards, Data hazard e Branch hazards.

## Conclusão

Nesta atividade refletimos sobre os processamentos paralelos e concluímos que duas técnicas são frequentemente utilizadas para a execução de instruções em paralelo:

- Execução encadeada de instruções
- Execução superescalar de instruções

Concluimos também que pipelines apresentam um problema conhecido como hazards e acontecem devido a interdependência entre as instruções e os dados utilizados pelas mesmas, os recursos de hardware necessários para a execução das mesmas e por fim pela dependência entre as instruções e o fluxo das instruções. Temos três tipos de hazards (structural hazards, data hazard e branch hazards).

Detetar e solucionar os hazards dos pipelines é um dos maiores problemas na construção dos mesmos, onde a solução mais simples pode ser esvaziar o pipeline antes da execução da instruções seguinte.

**Avaliação**

Este conteúdo será avaliado na avaliação sumativa da Unidade 1 que tem o peso de 5%.

**RESUMO DA UNIDADE**

Nesta unidade, aprendemos sobre seção do processamento e controlo do processador, formas de implementar a unidade de controlo, a linguagem de descrição e transferência de registo e operações internas no computador.

## Avaliação da Unidade

Para a avaliação desta unidade recomendamos o agrupamento dos estudantes em grupo de no máximo dois elementos e depois responder às questões abaixo:

Que os papéis são desempenhados pelos registadores dos processadores?

Porque um papeline de instruções de dois estágios dificilmente diminuirá o tempo do ciclo da instrução pela metade, quando comparado ao sistema do papeline?

Liste e explique resumidamente várias formas em que um papeline de instruções pode lidar com instrução de desvio condicionais.

Explique a diferença entre sequência de escrita e a sequência de tempo de uma instrução.

Qual a relação entre instruções e micro-operações.

Qual a função geral de uma unidade de controlo do processador.

Defina processador nos três passos que levam à caracterização da unidade de controlo.

Que tarefas básicas são efetuadas por uma unidade de controlo?

Enumere algumas aplicações comuns de microprogramação.

**Instruções:**

Organizar a turma em grupos de dois elementos e enviar os detalhes do grupo ao(à) instrutor(a) da disciplina através de correio electrónico.

Recorra sempre às pequisas na internet para o auxiliar a responder as questões.

Responda às questões num documento texto e envie-o ao (à) instrutor (a) da disciplina através do correio electrónico.

**Critérios de Avaliação**

Esta avaliação tem o peso de 5%.

**Comentários**

Caso necessitar de algum esclarecimento sinta-se livre para interagir com o (a) instrutor (a) através do correio electrónico, facebook, ferramentas do twitter ou Googledrive. O (a) instrutor (a) também irá comunicar consigo periodicamente fazendo comentários sobre o seu trabalho através de ferramentas como, correio electrónico, facebook. Estas ferramentas de comunicação irão ajudá-lo (a) a completar o seu trabalho e esclarecer as suas dúvidas.

Dê-nos as suas sugestões e / ou recomendações sobre a forma como o conteúdo desta unidade pode ser melhorado.

**Leituras e outros Recursos**

As leituras e outros recursos desta unidade encontram-se na lista de Leituras e Outros Recursos do curso.

# Unidade 2. Multiprocessamento

## Introdução à Unidade

Uma forma lógica de aumentar o desempenho de uma arquitectura é adicionando vários processadores. Teoricamente, a junção do N processadores pode conduzir a uma melhoria do desempenho em N vezes, atingindo uma capacidade de processamento superior a qualquer sistema uniprocessador.

Os processadores modernos são escalavéis porque o desempenho do sistema pode ser melhorado adicionando mais unidades de processamento. Esta técnica é muito utilizada em servidores de ficheiros e de bases de dados. Podem ser utilizados para executar uma só aplicação (por exemplo: simulação do tempo) ou para suportar a carga de vários utilizadores (p. ex. servidores Web).

## Objetivos da Unidade

Após a conclusão desta unidade, você deverá ser capaz de:

- Classificar os sistemas de multiprocessamento;
- Descrever  o multiprocessamento;
- Apontar as vantagens e desvantagens da taxonomia de Flynn
- Identificar as razões para um processador poder suportar a execução simultânea de programas.

**TERMOS-CHAVE**

**Lei de Amdahl's:** É utilizada para encontrar a máxima melhora esperada para um sistema em geral quando uma única parte do mesmo é melhorada.

**Multicore:** consiste em colocar dois ou mais núcleos de processamento (cores) no interior de um único chip. Estes dois ou mais núcleos são responsáveis por dividir as tarefas entre si, ou seja, permitem trabalhar em um ambiente multitarefa.

**Multiprocessador:** É a capacidade de um sistema operativo executar simultaneamente dois ou mais processos. Pressupõe a existência de dois ou mais processadores.

## Atividade 1 – Classificação dos sistemas (Taxonomia de Flynn)

**Introdução**

Taxonomia de Flynn baseia-se no fato de um computador executar uma sequência de instruções de dados, diferenciando fluxo de instruções e o fluxo de dados.

Abrange quatro classes de arquiteturas de computadores:

- SISD (Single Instruction Single Data): Fluxo único de instruções sobre um único conjunto de dados.
- SIMD (Single Instruction Multiple Data): Fluxo único de instruções em múltiplos conjuntos de dados.
- MISD (Multiple Instruction Single Data): Fluxo múltiplo de instruções em um único conjunto de dados.
- MIMD (Multiple Instruction Multiple Data): Fluxo múltiplo de instruções sobre múltiplos conjuntos de dados.
- Nesta atividade, vamos detalhar e exercitar o uso da taxonomia de Flynn.

**Detalhes da atividade**

Em 1972, Michael Flynn deu origem a uma taxonomia de hardware baseado em dois princípios:

**Número de fluxo de instruções**

**Número de fluxos de dados**

Um fluxo de instruções equivale a uma sequência de instruções executadas (em um processador) sobre um fluxo de dados aos quais estas instruções estão relacionadas.

Baseando-se na possível unicidade e multiplicidade de fluxos de dados e instruções, classificou as arquiteturas de computadores em quatro classes: SISD - Single Instruction Single Data – Um só fluxo de instruções e de dados, Corresponde ao modelo tradicional ou de Von Neumann, Um processador executa sequencialmente um conjunto de instruções sobre um conjunto de dados, A Figura 2.1 ilustra esta classe.

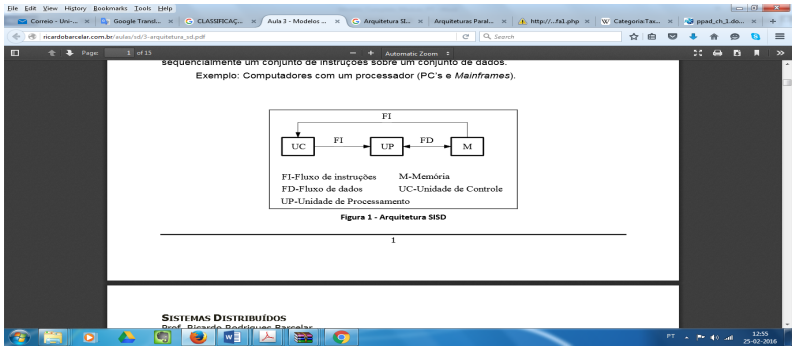



Figura 2.1: Arquitetura SISD

Fonte: http://ricardobarcelar.com.br/aulas/sd/3-arquitetura_sd.pdf consultado em 25-02-2016

SIMD - Single Instruction Multiple Data – Um só fluxo de instruções processa vários fluxos de dados.

Envolve múltiplos processadores (escravos) sob o controle de uma única unidade de controlo (mestre) executa simultaneamente a mesma instrução em diversos conjuntos de dados, Ilustrado na Figura 2.2.

São utilizadas para manipulação de matrizes e processamento de imagens.

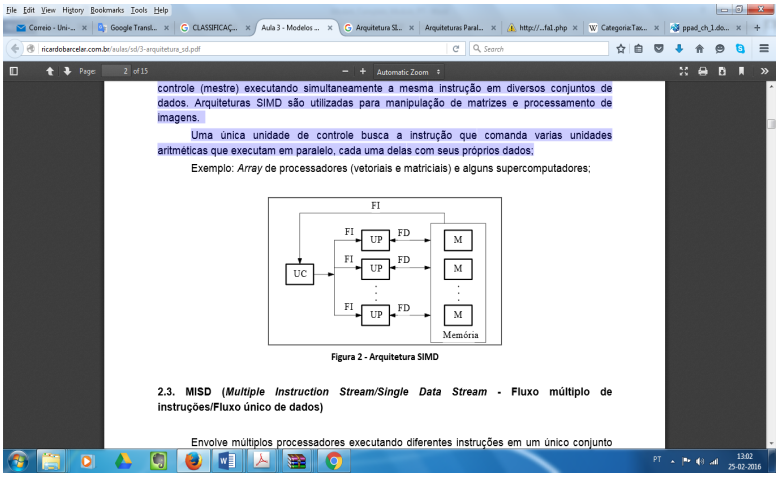



Figura 2.2: Arquitetura SIMD

Fonte: http://ricardobarcelar.com.br/aulas/sd/3-arquitetura_sd.pdf consultado em 25-02-2016

MISD - Multiple Instruction Stream/Single Data Stream - Envolve conjuntos de processadores a executar diferentes instruções em um único conjunto de dados, conforme ilustrado na Figura 2.3. Geralmente, nenhuma arquitetura é classificada como MISD, isto é, não existem representantes desta categoria. Alguns autores consideram arquiteturas pipeline.

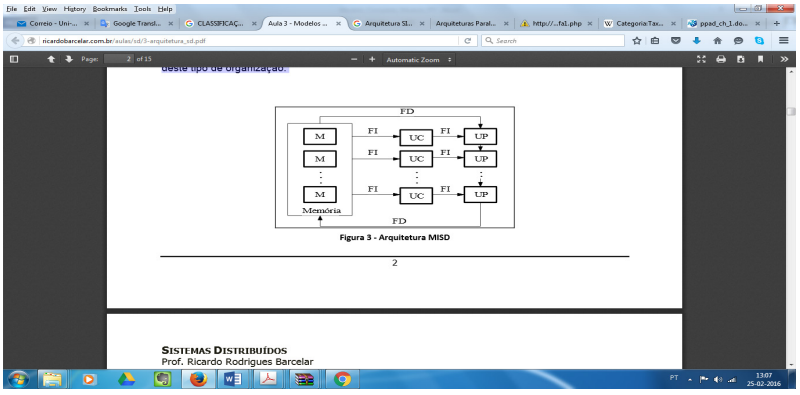



Figura 2.3: Arquitetura MISD

Fonte: http://ricardobarcelar.com.br/aulas/sd/3-arquitetura_sd.pdf consultado em 25-02-2016

MIMD - Multiple Instruction Multiple Data – Vários fluxos de instruções processam vários fluxos de dados, Figura 2.4.

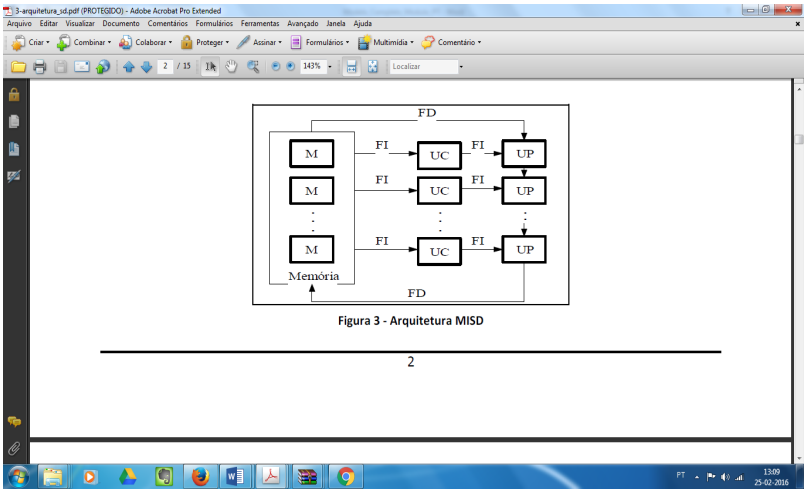



Figura 2.4: Arquitetura MIMD

Fonte: http://ricardobarcelar.com.br/aulas/sd/3-arquitetura_sd.pdf consultado em 25-02-2016

MIMD Tem prevalecido por ser mais flexível e poder ser desenvolvido com base em processadores comerciais. Esta categoria divide-se em duas categorias por tipo de memória, conforme ilustrado na Figura 2.5:

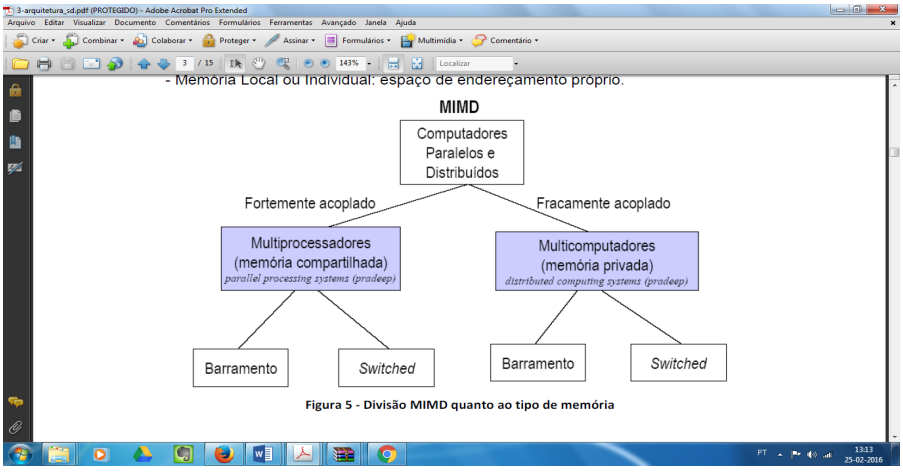



Figura 2.5: Divisão MIMD quanto ao tipo de memória

Fonte: http://ricardobarcelar.com.br/aulas/sd/3-arquitetura_sd.pdf consultado em 25-02-2016

**Memória partilhada centralizada:**

Possui um só espaço de endereçamento partilhado por todos os processadores;

Primitiva para sincronizar os acessos às zonas de memória partilhada;

Velocidade de acesso à memória pode ser uniforme (UMA).

**Memória distribuída:**

Cada processador possui o seu espaço de endereçamento;

Existem primitivas para o envio de informação entre processadores;

Sistemas híbridos de memória partilhada e distribuída

Velocidade de acesso à memória não uniforme (NUMA) variando em função do endereço acedido.

Com base na leitura do texto responda as seguintes questões:

Utilizando a classificação de Flynn, faça uma análise comparativa entre MIMD versus SIMD.

Enumere as limitações da classificação de Flynn.

A fim de acrescentar novas arquiteturas paralelas surgidas, sem descartar a classificação de Flynn (visto que esta é muito difundida), foi proposta uma classificação mais completa, e que permite apresentar uma visão geral dos estilos de organização para computadores paralelos da atualidade. Quem propôs este modelo?

**Resposta à questão N° 1**

Ambos os tipos de organizações de computadores apresentam vantagens e inconveniências.

As Arquiteturas SIMD, por apresentarem fluxo único de instruções, oferecem facilidades para a programação e depuração de programas paralelos. Além disso, seus elementos de processamento são simples, pois são destinados aos pequenos cálculos. Por outro lado, arquiteturas MIMD apresentam grande flexibilidade para a execução de algoritmos paralelos (arquiteturas SIMD usualmente destinam-se a processamento de propósito específico), e apresentam bom desempenho em virtude de seus elementos de processamento serem assíncronos.

**Resposta à questão N° 2**

A classificação de Flynn apresenta alguns problemas. Ela não é abrangente o suficiente para incluir alguns computadores modernos (por exemplo, processadores vetoriais e máquinas de fluxo de dados), falhando também, no que concerne à extensibilidade da classificação.

Outro inconveniente desta classificação é a falta de hierarquia. A classificação MIMD, por exemplo, engloba quase todas as arquiteturas paralelas sem apresentar sub-níveis. No entanto, apesar de antiga (proposta em 1972), a classificação de Flynn é bastante concisa e a mais utilizada.

**Resposta à questão Nº 3**

Duncan

Conclusão

Nesta atividade, pudemos constatar que Flynn deu origem a uma taxonomia de hardware com o seu nome baseado em dois princípios: Número de fluxo de instruções e Número de fluxos de dados e baseou na possível unicidade e multiplicidade de fluxos de dados e instruções para classificar a arquiteturas de computadores em quatro classes: SISD, SIMD, MISD,MIMD.

**Avaliação**

Este conteúdo será avaliado na avaliação sumativa da Unidade 2 que tem o peso de 5%.

## Atividade 2.2 - Lei de Amdahl

**Introdução**

Um computador grosso modo, nada mais é que uma máquina que realiza vários cálculos. Computação/Arquitetura paralela é uma forma de realizar esses cálculos simultaneamente, utilizando o princípio que um dado problema pode ser dividido em problemas menores que são resolvidos concorrentemente, ou seja, em paralelo.

Em geral, a necessidade de utilização dos computadores cresce cada vez mais, além do fato que o hardware possui limitações físicas que impedem o aumento da freqParei aquiuência de processamento.

Nesta atividade vamos analisar o quão mais rápido uma tarefa será executado quando utilizamos um computador com alguma melhoria em relação ao computador atual, isto é, elucidar como calcular os ganhos em eficiência e o desempenho quando alteramos algum componente ou dispositivo.

**Detalhes da atividade**

A Lei de Amdahl concentra-se no conceito de limites de ganho sobre o tamanho fixo da carga computacional (workload).

A Lei de Amdahl pode ser aplicada tanto na otimização de programas seriais como paralelos. Se olharmos pelo ponto de vista da otimização de programas seriais, a lei diz que, mantendo fixo o tamanho do problema, o speedup será limitado pela fração do problema que não puder ser otimizado e pelo seu respetivo tempo de execução.

Do ponto de vista da otimização por paralelização, a lei diz que se mantivermos fixo o tamanho do problema, o speedup será limitado pela fração do programa que não puder ser paralelizada, denominada, fração serial. Por exemplo, se tivermos um programa que é executado de forma serial em 20 horas, e se uma parte dele, que dura 1 hora, não puder ser paralelizada e as 19 horas restantes puderem ser paralelizadas em 95%, utilizando 4096 processadores alcançamos o limite de speedup de 20x. Assim, mesmo que tenhamos mais que 4096 processadores disponíveis, o tempo de execução será limitado em 1 hora que corresponde à parte serial do programa que não pode ser paralelizada.

Assim, dada uma carga computacional (workload) que leva 20 horas de processamento, duplicando a carga e mantendo-se os 95% de paralelização o speedup será de 40x. Isto é, torna-se desprezível a parte constante (serial) do problema conforme aumenta o volume de dados.

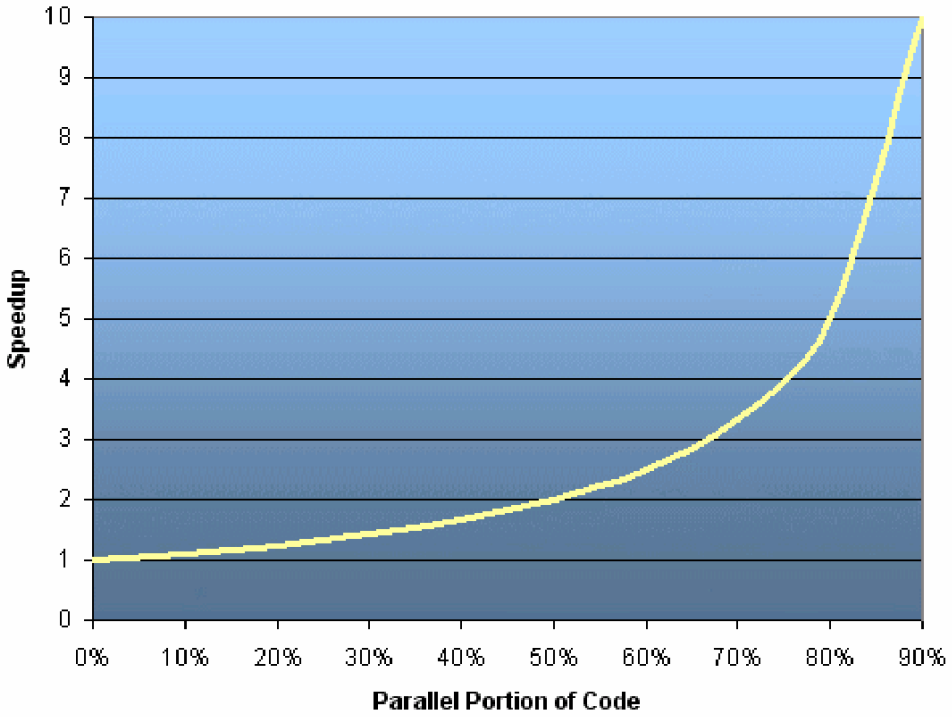



Figura 2.6:Paralelização vs. speedup teórico segundo a lei Amdahl.

**Fonte: (Breshears, 2009)**

A figura 2.6 mostra, segundo a lei de Amdahl, o comportamento do speedup conforme aumenta o número de recursos de processamento disponíveis.

Assim, se 5% do tempo de execução de um programa não puder ser paralelizado, o limite de speedup será de 20x a partir da utilização de 1024 processadores.

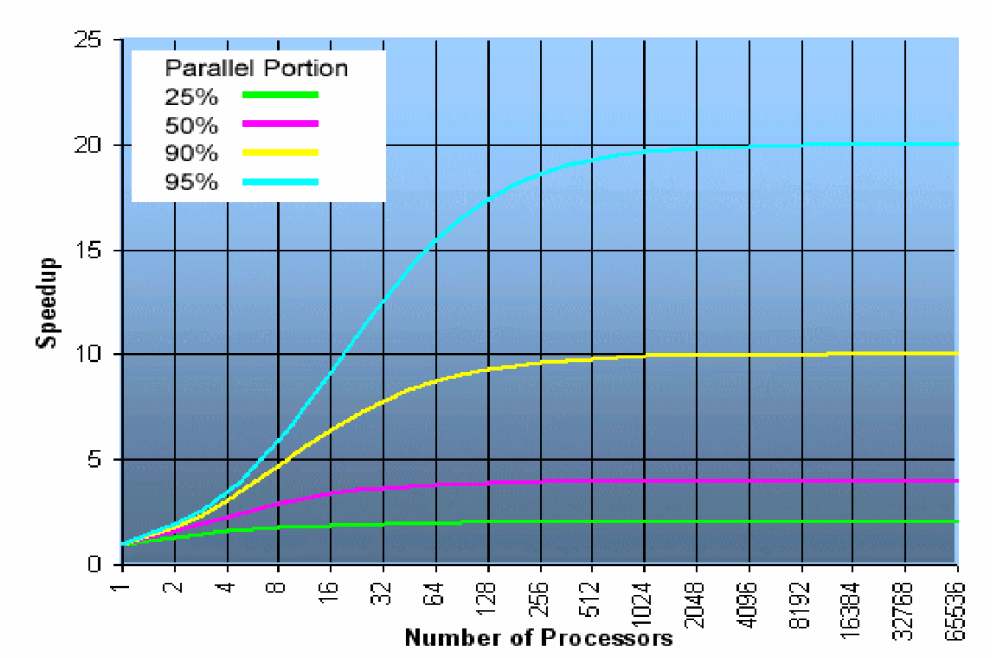



Figura 2.7: Exemplo de curvas de speedup

**Fonte: (Breshears, 2009)**

A lei de Amdahl define o speedup que pode ser obtido utilizando uma característica particular.

Suponha que seja possível fazer uma modificação em um computador que irá melhorar o desempenho quando ele é utilizado. O speedup é a taxa conforme a figura 2.8:

$$S(p) = \frac{TempoSemMelhora}{TempoComMelhora}$$

Figura 2.8: Limites de speedup segundo a lei de Amdahl

**Fonte: (Breshears, 2009)**

O speedup nos diz o quão mais rápido uma tarefa irá ser executada utilizando um computador com alguma melhoria em relação ao computador original, (Patterson et all, 2007).

Suponha que uma melhoria seja a possibilidade de executar em paralelo. Uma formulação da lei de Amdahl é demonstrada na figura 2.9, onde pctPar é a percentagem de tempo de execução que irá executar em paralelo e p é o número de núcleos nos quais a aplicação irá executar em paralelo (Breshears, 2009):

$$S(p) = \frac{1}{(1 - pctPar) + \frac{pctPar}{p}}$$

Figura 2.9: Demonstração da lei de Amdahl

**Fonte: (Breshears, 2009)**

Observamos que o speedup é proporcional à fração de tempo em que a melhoria pode ser utilizada e ao número de processadores que executarão essa melhoria. A Figura 2.10 apresenta exemplos de curvas de speedup. O speedup perfeito ocorre quando o speedup calculado é igual ao número de núcleos (linha continua). Para uma aplicação ter uma boa escalabilidade, o speedup deveria aumentar próximo ou na mesma proporção que novos núcleos são adicionados (linha tracejada), ou seja, se o número de núcleos é duplicado, o speedup também deve aumentar na mesma proporção. Se o speedup de uma aplicação não acompanha essa proporção (linha pontilhada), conforme a figura 2.10 dizemos que a aplicação não foi bem escala (Breshears, 2009).

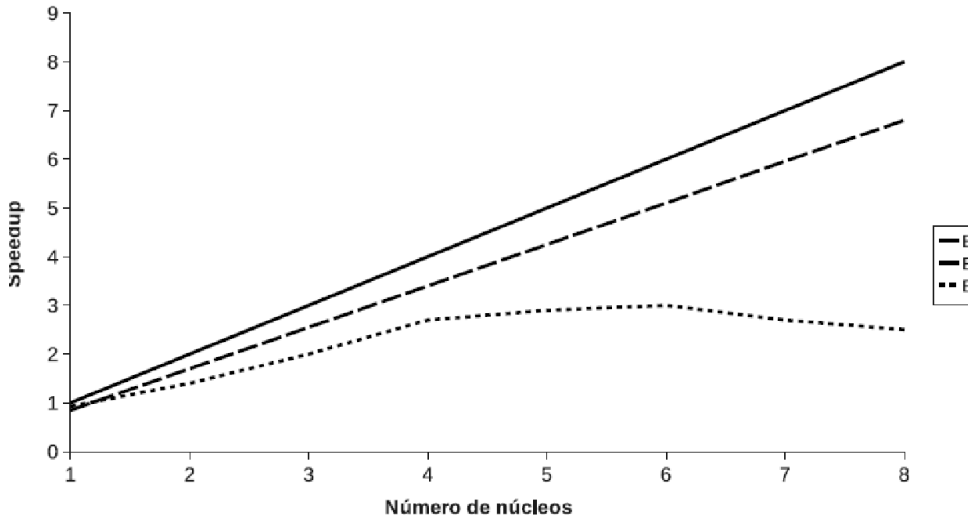



Figura 2.10: Exemplo de curvas de speedup

**Fonte: (BRESHEARS, 2009)**

Outra métrica relacionada ao speedup é a eficiência.

Enquanto o speedup nos dá uma métrica para determinar quão mais rápido é uma aplicação paralela em comparação com sua implementação sequencial, a eficiência nos diz o quão bem os recursos computacionais estão sendo utilizados. Para calcular a eficiência de uma execução paralela, deve-se dividir o speedup pelo número de núcleos utilizados. Esse número é expresso em forma de percentagem (Breshears, 2009). Por exemplo se temos um speedup de valor 48 a executar numa máquina com 64 núcleos, a eficiência alcançada é 75 por cento, ou seja, 48/64 = 0.75. Isso significa que, em média, durante o curso da execução, cada um dos núcleos está inativo durante 25 por cento do tempo. O cálculo da eficiência é apresentado na Figura 2.11:

$$E(p) = \frac{S(p)}{p} = \frac{T(1)}{pT(p)}$$

Figura 2.11: Fórmula para cálculo da eficiência

**Fonte: (BRESHEARS, 2009)**

Após leitura, em detalhe, do texto e interpretação dos gráficos apresentados na atividade 2.2, responda às seguintes questões:

Quais as limitações da lei de Amdahl;

Se um computador executa um programa P em 100 segundos, onde 30% das operações são acessos à memória com tempo médio de acesso de 80ms e 60% são operações de ponto flutuante.

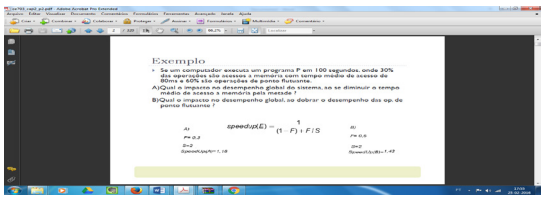
Quais os impactos no desempenho global do sistema se diminuírmos o tempo médio de acesso a memória pela metade?

Qual o impacto no desempenho global, ao duplicarmos o desempenho das operações de ponto flutuante?

**Resposta à questão Nº 1**

A grande limitação da lei de Amdahl é que ela só se aplica aos casos em que o tamanho do problema está corrigido. Na prática, como mais recursos de computação se tornam disponíveis, eles tendem a acostumar-se com problemas maiores (maiores conjuntos de dados), e o tempo gasto na parte paralelizável geralmente cresce muito mais rápido do que o trabalho inerentemente sequencial. Neste caso, a lei de Gustafson permite fazer uma avaliação mais realista do desempenho paralelo.

**Resposta à questão N° 2.a**



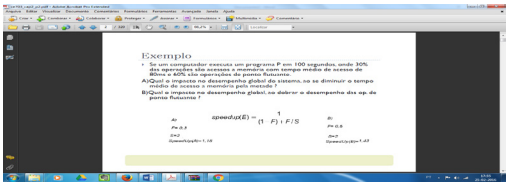

A)

F= 0,3

S=2

SpeedUp(A)=1,18

**Resposta à questão N° 2.b**




B)

F= 0,6

S=2

SpeedUp(B)=1,43

## Conclusão

Nesta atividade, analisamos como a lei de Amdahl pode ser utilizada para determinar o quão rápido uma tarefa será executada quando implementarmos alguma melhoria em relação ao computador atual.

Do ponto de vista da otimização por paralelização, a lei diz que se mantivermos fixo o tamanho do problema, o speedup será limitado pela fração do programa que não puder ser paralelizada.A lei de Amdahl define o speedup.

**Avaliação**

Este conteúdo será avaliado na avaliação da Unidade 2 que tem o peso de 5%.

## Atividade 3 – Multi-core e multi-processador

**Introdução**

Desde a última década, a evolução dos processadores acontece de forma surpreendente.

Isto deve-se principalmente ao aumento na densidade de integração de transistores em áreas cada vez menores. Neste nível de integração, é necessário explorar técnicas de paralelismo para efetivamente utilizarmos, ao máximo, o desempenho dos processadores.

Pipeline, superescalaridade, multithreading, são algumas técnicas aplicadas na exploração de paralelismo para aumento de desempenho.

Nesta atividade, serão analizados os sistemas Multi-core e Multi-processador.

**Detalhes da atividade**

**Multicore ou Multinúcleo**

É o termo utilizado para descrever a utilização de mais de um núcleo em um processador, cada um desses cores tem sua cache independente e executam programas em paralelo sem interferir no processamento do outro.

Sua lógica é baseada ns subdivisão de um único processador, como era utilizado em anos passados, em processadores menores para combater o consumo de energia e dissipação do calor de um processador à medida que estes eram desenvolvidos com mais sistemas embarcados (Alves, 2015).

**Arquitetura Multicore**

Uma das arquiteturas utilizadas pelos multicores é conhecida como Simultaneous Multiprocessing (SMP), que é ter múltiplos cores do processador compartilhando as demais memórias de níveis inferior (Douglas, 2015). Esta arquitetura favorece a utilização de múltiplos processos e de threads fazendo assim com que aumentem o troughput das máquinas.

A arquitetura em si utilizada pelos multicores não  se diferencia da ideologia da arquitetura de Von Neumann que é utilizada nos single cores, os multi-nucleos mantêm a mesma interação processador-memória-dispositivos entrada/saída, a logica de seu funcionamento será a mesma, a diferença entre os dois modelos é a quantidade de núcleos que atuam na CPU do computador utilizando uma memória compartilhada.

Os multicores sugiram com o intuito de manter o alto desempenho de processamento das máquinas sem ter que investir em seus arrefecimentos, devido ao aquecimento causado pela alta velocidade e frequência que as instruções eram processadas pela CPU. Com o surgimento dessa nova linha de montagem de processadores programas distintos podem ser executados simultaneamente permitindo a utilização de um novo conceito que é a programação em paralelo, que aproveita, ao máximo, as vantagens disponíveis das threads.

**Desempenho Multicores**

A utilização de mais de um núcleo nos multicores faz com que eles aumentem a velocidade de processamento por pulso de relógio, compartilhem o processamento de dados entre os cores e otimizem o desempenho de certas tarefas e diminuam o período de execução. O tempo de execução da CPU para um programa é dado pela razão dos seus ciclos de relógio por velocidade de relógio, portanto, aumentando a velocidade de relógio consequentemente aumenta o tempo de execução. A arquitetura dos multicores proporciona um aumento no desempenho da execução de dados, dois dos fatores que ajudam nesse aumento é a utilização de Multi-threading e Programação Paralela, a seguir é exibido como é feita a utilização das duas ideologias.

**Multi-threading (MT):**

O conceito de threads está em fazer um escalonamento de programas que estejam em execução para serem processados pelo processador. Como os multicores trabalham com mais de um núcleo em seus processadores, esta atividade é definida por multi-threads e tem o escalonamento dos programas distribuídos entre os núcleos dos processadores. Para melhorar o funcionamento dos multicores foi desenvolvido uma técnica chamada de simultaneous multithreading (SMT) e tem por objetivo executar várias threads simultaneamente evitando os desperdícios horizontais do processador (unidades funcionais não utilizadas), ele interpreta que existem vários programas em execução, e com a maximização da utilização das unidades funcionais das threads aumenta o CPI dos programas e consequentemente o desempenho do processador (Alves, 2015). O SMT permite que as threads sejam processadas em paralelo dentro de cada núcleo, o processador executa normalmente as instruções de threads a cada ciclo de processamento, quando este fica em estado de espera executa em paralelo outra thread. Exemplo: Enquanto o processador espera que uma execução de ponto flutuante seja finalizada, outras instruções podem ser executadas em threads inteiros.

**Programação paralela:**

O conceito de programação em paralelo é fazer com que um programa seja visto como um conjunto de partes em que possam ser resolvidas de forma concorrente. A sua construção é feita de forma sequencial mas é feita de tal modo que possa ser executada de forma paralela em diferentes processadores. A ilustração é dada pela figura 2.12.

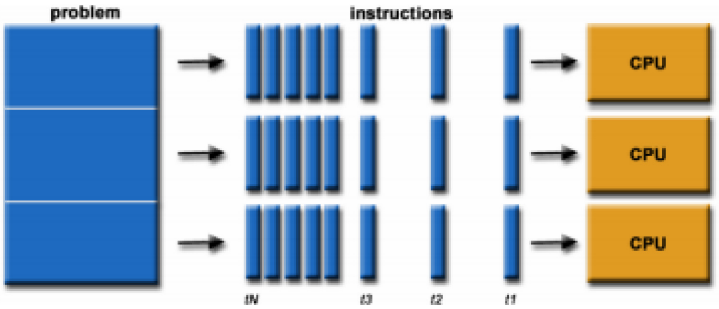



Figura 2.12: Programação paralela

**Fonte: Adpado pelo autor**

Programas de computador paralelos logicamente são feitos de forma diferente aos sequenciais e de forma mais complexa, pois a comunicação e a sincronização das sub-tarefas torna-se um empecilho para a elaboração de bons programas. O programador deve ter o cuidado de saber quais e quando as instruções vão ser executadas em paralelo, em que ordem e definir o nível de paralelismo (Alves, 2015).

Atualmente existem 4 formas de programação paralela: em bit, instrução, dado ou de tarefa, que são determinadas por:

Em bits: deve-se ao aumento da quantidade de bits de uma instrução (palavra), aumentando os bits diminui-se a quantidade de instrução.

Instrução: reordenar as instruções, de forma que não altere o seu resultado, para que estas aproveitem ao máximo os estágios dispostos pelo pipeline (tratamento de hazards).

Dado: utilização de loops de forma inteligente e correta em que um laço não dependa de dados de laços anteriores que ainda estão por ser calculados.

Tarefa: Evitar que vários cálculos sejam executados por um mesmo conjunto de dados.

Os principais motivos para utilizarmos a programação paralela são resolver processos mais complexos com maior dimensão e reduzir o tempo para solucionar um problema. Ela induz a falsa crença que com a sua utilização, o desempenho de execução das suas instruções possa duplicar, porém uma pequena parcela do programa que não pode ser paralelizada limitará este desempenho, por isso é sempre bom procurar o paralelismo nos códigos para se obter o máximo de benefícios que esta técnica proporciona.

**Multiprocessamento**

É a capacidade de um sistema operativo executar simultaneamente dois ou mais processos. Pressupõe a existência de dois ou mais processadores. Difere da multitarefa, pois esta simula a simultaniedade, utilizando-se de vários recursos, sendo o principal o compartilhamento de tempo de utilização do processador entre vários processos (Wikipedia).

Hoje, multiprocessadores são comumente encontrados na mesma placa física e conectados através de uma interface de comunicação de alta velocidade conforme a Figura 2.13.

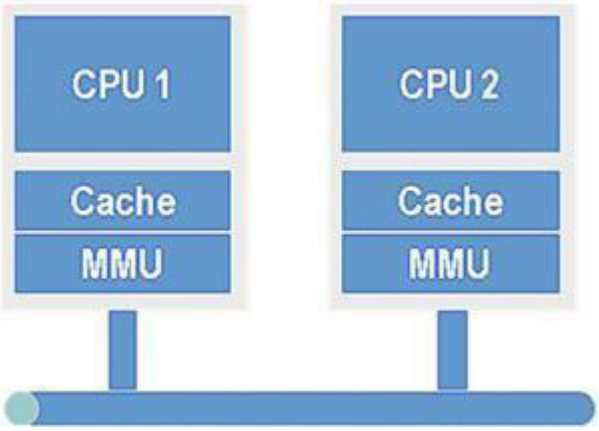



Figura 2.13: Sistema de multiprocessadores

Fonte: http://www.ni.com/white-paper/6964/pt/#toc1 consultado em 25-02-2016

**Características de Multiprocessamento**

Segundo Wikipedia, Um sistema multiprocessador possui as seguintes características:

Envolve dois ou mais processadores físicos (sejam processadores separados ou múltiplos núcleos encapsulados no mesmo chip) ou lógicos (processador(es) com a tecnologia HyperThreading da Intel) com o mesmo poder de computação e cada um capaz de executar processos autonomamente. Isto implica que não há nenhuma unidade "central" de controlo; cada processador possui sua própria unidade de controlo. Assim, efetivamente, a lógica de controle é distribuído pelo sistema. Os processadores compartilham um único espaço de endereçamento de memória. O sistema de hardware é como um todo gerido por um único sistema operacional.

O sistema operativo com suporte ao multiprocessamento deve ser capaz de:

Suportar multitarefa;

Manter múltiplas filas de processos, uma para cada processador.

**Arquitetura de Multiprocessamento**

Sistemas multiprocessador podem ser de dois tipos:

Multiprocessamento simétrico (SMP): os processadores compartilham a mesma memória, embora possam ter caches separadas. O sistema operacional deve estar preparado para trabalhar com coerência de caches e, principalmente, evitar condições de corrida na memória principal.

Acesso não uniforme à memória (NUMA): a cada processador é associado uma base de memória. Nesse caso, o sistema operacional trata cada banco separadamente, pois cada banco tem um custo de acesso diferente, dependendo de qual o processador a que está associado e onde está sendo executado o processo que tenta aceder à memória.

## Conclusão

Sistemas Multiprocessadores são menos complexos que sistemas Multicore porque são CPUs de chips únicos e essenciais conectados juntos. A desvantagem de sistemas multiprocessadores é que estes são mais caros pois requerem vários chips que uma solução de chip único.

Com o surgimento dos multicores puderam ser criadas novas formas de manipulação dos dados por parte dos computadores, proporcionando uma vasta área de pesquisa para o desenvolvimento de processos que melhorem cada vez mais o processamento das máquinas.

Esta tecnologia proporcionou uma nova visão de rumo para como os profissionais da área de TI devem se preparar para lidar com os novos meios de desenvolvimento de programas, o que causa uma ótima visão para o surgimento de softwares e sistemas de melhores qualidades.

**Avaliação**

Este conteúdo será avaliado na avaliação da Unidade 2 que tem o peso de 5%.

## RESUMO DA UNIDADE

Nesta unidade analisamos, a taxonomia de Flynn confirmamos que ela se baseia no fato de um computador executar uma sequência de instruções de dados, deferenciando o fluxo de instruções e o fluxo de dados. Abrange quatro classes de arquiteturas de computadores: SISD, SIMD, MISD e MIMD.

Analizamos ainda o quão rápido uma tarefa será executada quando utilizamos um computador com alguma melhoria em relação ao computador atual, isto é, analizamos como calcular os ganhos em eficiência e o desempenho quando alteramos algum componente ou dispositivo.

Vimos também que sistemas multiprocessadores são menos complexos que sistemas Multicore porque são CPUs de chips únicos e essenciais conectados juntos. A desvantagem de sistemas multiprocessadores é que estes são mais caros pois requerem vários chips que uma solução de chip único.

## Avaliação da Unidade

Responda às seguintes questões:

Apresente, de forma resumida, a diferença entre pipeline de instrução simples e superscalar.

Enumere várias razões para a migração dos projetistas para uma organização multicore em vez de aumentar o paralelismo dentro de um único processador.

Quais são as diferenças entre UMA e NUMA?

Relacione e defina de forma resumida três tipos de categorias de sistemas informáticos.

Para cada uma das questões a seguir escolha a resposta mas adequada:

De acordo com a taxonomia de Flynn, os sistemas multiprocessados e os embutidos pertencem à categoria:

- MIMD
- MISD
- SISD
- SIMD
- SIMS

Os processadores utilizam diferentes técnicas para acelerar a execução de instruções. Uma dessas técnicas envolve a divisão do ciclo de instruções em um determinado número de estágios consecutivos, possibilitando que cada estágio trabalhe simultaneamente em uma instrução diferente. Essa técnica chama-se:

- cache
- stacking
- pipelining
- turbo boost
- hyper-threading

Uma aplicação que apoia a operação de uma grande empresa faz muitos acessos à memória. Para saber se a política de atualização da cache do servidor dessa aplicação está adequada, verificou-se que a taxa de acertos (hit rate) da cache é de 97%. Considerando que foram feitos 300.000 acessos no total, que o tempo por acerto (tempo por hit) é de 70ns e que o tempo por falha (tempo por miss) é de 3000ns para este mesma cache, qual o tempo, em ns, de acesso efetivo?

- 42,86
- 157,9

- 2912,1
- 3000
- 3070

A taxonomia de Flynn utiliza duas dimensões independentes: instruções e dados. Essa taxonomia, registra, na arquitetura SIMD, que:

Uma única instrução é executada ao mesmo tempo sobre múltiplos dados.

Um único fluxo de instruções atua sobre um único fluxo de dados.

Cada unidade de processamento pode executar instruções diferentes e operar sobre fluxos de dados diferentes a cada momento.

Múltiplos fluxos de instruções atuam sobre um único fluxo de dados.

Múltiplas unidades de processamento executam múltiplas instruções simultaneamente e operam diversos fluxos de dados sobre cada uma dessas unidades.

**Instruções**

Organizar a turma em grupos de dois elementos e enviar os detalhes do grupo ao (à) instrutor (a) da disciplina através de correio electrónico.

Recorra sempre a pequisas na internet para o (a) auxiliar a responder às questões.

Responda às questões num documento texto e envie ao (à) instrutor(a) da disciplina através do correio electrónico.

**Avaliação**

A avaliação desta unidade tem o peso de 5%.

# Unidade 3: Programação em Baixo Nível

## Introdução à Unidade

Uma linguagem de programação pode ser definida como sendo um conjunto limitado de instruções (vocabulário), associado a um conjunto de regras (sintaxe) que define como as instruções podem ser associadas, ou seja, como se pode compor os programas para a resolução de um determinado problema.

As linguagens de programação podem ser classificadas em níveis de linguagem, sendo que as linguagens de nível mais baixo estão mais próximas da linguagem interpretada pelo processador e mais distantes das linguagens naturais.

## Objetivos da Unidade

Após a conclusão desta unidade, você deverá ser capaz de:

Apontar a estrutura de programas de baixo nível;

Identificar as limitações de arquiteturas de baixo nível;

Identificar a arquitetura que suporta as linguagens de baixo e alto nível e linguagem assembler;

**TERMOS-CHAVE**

**MPLAB:** É um software editor IDE (Integrated Development Environment) para gestores de projetos e ambiente de programação para desenvolvimento de aplicações e sistemas embutidos.

**MicroChip:** É propriamente, um circuito integrado (IC).

**PIC:** Controlador de interrupção que providencia um mecanismo para que dispositivos conectados obtenham atenção da UCP.

**Atividades de Aprendizagem**

## Atividade 3.1 - Estrutura de programas de baixo nível

**Introdução**

Linguagem de programação de baixo nível trata-se de uma linguagem que compreende as características da arquitetura do computador.

Utiliza apenas instruções do processador, para isso é necessário ter conhecimentos dos registradores da máquina. Nesse sentido, as linguagens de baixo nível estão diretamente relacionadas com a arquitetura do computador.

Segundo a Wikipedia, consuldado em Abril de 2016, as linguagens de baixo nível são divididas em duas categorias: primeira e segunda geração.

A primeira geração, é o código de máquina. É a única linguagem que um microprocessador entende nativamente. O código máquina não pode ser escrito ou lido por um editor de texto, e é raramente utilizado por pessoas.

Na segunda geração, temos a linguagem Assembly, embora não seja uma linguagem nativa do microprocessador, um programador que utiliza a linguagem Assembly deve ainda compreender as características da arquitetura do microprocessador.

Nesta atividade vamos aprofundar a estrutura de uma linguagem de baixo nível.

**Detalhes da atividade**

As linguagens de Baixo Nível estão voltadas para a máquina, ou seja, são escritas utilizando instruções do microprocessador. São genericamente chamadas de linguagens Assembly. Os programas escritos com Alto Nível geralmente podem ser convertidos com programas especiais para Baixo Nível.

Ela possui algumas vantagens e limitações que serão discutidas na atividade 3.2.

**Estrutura de um programa de baixo nível**

Um programa de baixo nível é tipicamente composto por pelo menos dois segmentos: um segmento de dados que define o espaço associado ao armazenamento das variáveis e constantes utilizadas pelo programa; e um segmento de instruções, onde o código do programa é armazenado. Além dessas duas seções, pode conter uma seção de definições, utilizadas na descrição dos programas e que não produzem nenhum efeito no código criado.

Por conveniência da leitura de código fonte, a seção de definições é tradicionalmente atribuida no início do código. Assim, quando o código for lido por um humano terá a noção do significado das constantes simbólicas utilizadas ao longo do programa. Com relação aos segmentos de dados e de instruções, não há um posicionamento fixo. Na prática, um programa pode ter vários segmentos associados.

Para o montador, o posicionamento dos diferentes trechos de programa no código fonte deve ser irrelevante. Na verdade, é necessário que o montador seja capaz de manipular representações simbólicas antes que elas tenham sido definidas. Considere o exemplo da Figura 3.1:

```
1       START     ADD.L     D0,D1
2                 JMP       NEXT
3       LOOP      ADD.L     #1,D1
4       NEXT      CLR.L     D5
5                 JMP       LOOP
```

Figura 3.1: Trexo de programa

**Fonte: Adaptado da Wikipédia**

Na linha 2 desse trecho de programa há uma referência a um símbolo, NEXT, cujo valor não foi definido - essa definição só acontecerá na linha 4.

**Há duas possibilidades de lidar com essas referências futuras:**

A primeira possibilidade é deixar uma lacuna reservada no código criado associada ao operando da instrução da linha 2. Posteriormente, quando houver uma definição desse valor - provavelmente quando chegarmos ao fim do ficheiro essa definição é encontrada – e essa lacuna é preenchida. Neste caso, seria possível criar o código de máquina que realiza um único passo (uma única leitura) sobre o ficheiro. Entretanto, haveria um maior custo na complexidade de implementação, que deveria manter referências a todas as lacunas que devem ser preenchidas no final.  A outra possibilidade, conceitualmente mais simples, é realizar o processo em dois passos. O primeiro passo simplesmente lê o ficheiro com o objetivo de criar a tabela de símbolos, ou seja, obter os valores associados a todas as constantes simbólicas definidas no programa. No segundo passo, fazíamos uma nova leitura sobre o ficheiro com o objetivo para criar o código máquina; nesse passo, a informação da tabela de símbolos criada no primeiro passo é utilizada.

**Após a leitura do detalhe da atividade 3.1 responda as seguintes questões:**

Comente a seguinte afirmação: “Numa linguagem de baixo nível, o programador sofre, pois a linguagem é muito mais voltada para os dispositivos (processador, microcontrolador, entre outros).” Se programar em baixo nível é muito complicado, porque não utilizamos sempre a linguagem de alto nível, dado que com ela é mais fácil programar e fazer manutenções?

**Resposta a questão N° 1**

Verdadeiro. Porque normalmente programar em baixo nível envolve números e letras que nada mais são que instruções diretas ao dispositivo. Por exemplo, a instrução “MOV 1, R0” diz para mover o valor “1” para o registrador “0” de um determinado processador. Como vemos, o programador tem que entender não só da linguagem em si, mas toda a arquitetura do dispositivo que ele irá trabalhar.

**Resposta a questão Nº 2**

Não utilizamos sempre as linguagem de alto nível porque ás vezes, pela arquitetura do que se quer implementar essa opção não está disponível, pois a performance do dispositivo pode ser prejudicada se for utilizada uma linguagem de alto nível. Isso também tem a ver com memória: uma linguagem de alto nível, normalmente ocupa mais memória do que uma de baixo nível e aí o fator custo pode ser um obstáculo.

## Conclusão

As nossas principais conclusões nesta atividade são que as linguagens de baixo nível são divididas em duas categorias: primeira e segunda geração. Na primeira geração, temos o código de máquina enquanto na segunda, temos a linguagem Assembly. Um programa de baixo nível é tipicamente composto por pelo menos dois segmentos: um segmento de dados que define o espaço associado ao armazenamento das variáveis e constantes utilizadas pelo programa; e um segmento de instruções, onde o código do programa é armazenado Além dessas duas, pode conter uma seção de definições.

**Avaliação**

Este conteúdo será avaliado na avaliação sumativa da Unidade 3 que tem o peso de 5%.

## Atividade 3.2 – Vantagens e limitações de arquiteturas de baixo nível

**Introdução**

O computador só entende a linguagem conhecida como binário ou máquina, que consiste em zeros e uns. Ou seja, só utiliza 0 e 1 para codificar qualquer ação.  A linguagem assembly é, o nível mais baixo em que se pode programar com alguma comodidade. A tradução da linguagem assembly de cada microprocessador para o código máquina correspondente pode ser feita à mão, mediante uma tabela de conversão, mas normalmente é feita recorrendo a um assemblador, ferramenta que, na maior parte dos casos, é oferecida pelo fabricante do microprocessador. Nesta atividade apresentaremos as vantagens e desvantagens de programar em arquiteturas de baixo nível.

**Detalhes da atividade**

A principal vantagem de programar em linguagem de baixo nível é poder optimizar o código de modo a aproveitar ao máximo as caracteristicas particulares do hardware onde vai ser executado, conseguindo assim melhores resultados, quer em tempo de execucão quer em tamanho de código criado.

Outra vantagem é a existência de assembladores para todos os microprocessadores, muitas vezes oferecidos pelos fabricantes, pelo que é sempre possível programar em baixo nível, qualquer que seja o microprocessador escolhido. O mesmo não acontece com linguagens de alto nível, onde nem sempre é possível encontrar um compilador adequado para um determinado microprocessador.

**Podemos sumarizar as vantagens como:**

Programas são executados com maior velocidade e ocupam menos espaço na memória;

Permite criar ações de alta complexidade, impossíveis ou difíceis de se realizar em linguagens de alto nível;

O seu conhecimento  possibilita a programação em outras linguagens;

É recomendável quando o tempo é um fator crítico como por exemplo: medições de tempo que exigem boa performance;

É de fácil compreensão desde que tenhamos algum conhecimento em conceitos de hardware e seus dialetos.

A principal desvantagem de uma linguagem de baixo nível é a grande desproporção que existe entre a complexidade do seu conjunto de instruções e as tarefas que o microprocessador normalmente é chamado a executar. Esta desproporção obriga o programador a decompor cada tarefa num conjunto de operações elementares que, além de ser um processo demorado e sujeito a erros, não ajuda a manter o código estruturado.

Outra desvantagem é a necessidade de conhecer, em detalhe, o modelo de programação do microprocessador, nomeadamente no que se refere aos registos de trabalho disponíveis, registos privilegiados ou especiais e registo de estado. Como consequência desta dependência relativamente aos detalhes internos de um microprocessador, a portabilidade dos programas é muito reduzida.

**Podemos sumarizar as desvantagens como:**

**Possibilitar pouca portabilidade;**

Difícil ler, aprender, entender e fazer manutenção. Um programa escrito em baixo nível não é muito legível, por isso deve ser muito bem documentado.

Consumir muito tempo ao programador;

Poder ser necessário desenvolver um programa para cada máquina;

Não existir rotinas pré-definidas, o programador deverá desenvolver suas próprias rotinas;

Apresentar um número muito reduzido de instruções, do tipo operações de movimentação de dados em memória, para registros e para memórias, e operações lógicas e aritméticas simples.

Estas instruções são de baixa expressividade, o programador deve programar num nível de detalhamento muito maior para fazer a mesma coisa que em um programa escrito em linguagem de alto nível.

Programador utilizar diretamente os recursos do processador e memória, deve conhecer muito bem a máquina sobre a qual está a programar.

Após a leitura cuidada do detalhe da atividade 3.2 responda às seguintes questões:

Porquê uma linguagem de baixo nível tem normalmente melhor performance do que uma linguagem de alto nível?

**Resposta à questão Nº 1**

Por ser uma linguagem mais próxima ao dispositivo, são necessárias menos conversões dessa linguagem para a "linguagem de máquina" do dispositivo.

Quando escrevemos um programa utilizando uma linguagem de alto nível, muitas conversões/traduções são necessárias para alcançar a "linguagem máquina", como essa criação é feita de forma automática, o código resultante acaba sendo maior do que se fosse escrito diretamente por um programador utilizando a linguagem de baixo nível.

## Conclusão

A tendência atual é a favor de uma programação mista, utilizando principalmente linguagens de mais alto nível (C em particular) e recorrendo à linguagem assembly apenas em rotinas onde a eficiência do código seja o objectivo principal a atingir. Esta tendência explica-se por três motivos:

A pressão do mercado obriga a encurtar o tempo de desenvolvimento e a aumentar a facilidade de manutenção do código.

Existem actualmente compiladores de C para a maioria dos microprocessadores, alguns até de domínio público.

Os avanços na microelectrónica permitem que a rapidez de execução se consiga facilmente por aumento da frequência de funcionamento.

Quando se está a estudar o funcionamento interno de um microprocessador as arquiteturas de baixo nivél são as mais adequadas.

**Avaliação**

Este conteúdo será avaliado na avaliação sumativa da Unidade 3 que terá o peso de 5%.

## Atividade 3 – Tradução de Linguagem de Programação e linguagem Assembler

### Introdução

A maioria dos programas passa por quatro etapas de transformação: inicia no código-fonte armazenado no computador e chega a etapa em que este código é executado na máquina. A figura 3.2, ilustra essas etapas de tradução, alguns sistemas combinam essas etapas por forma a agilizar o processo.

Nesta atividade vamos identificar as etapas de tradução de linguagem de alto nivél para baixo nível e introduzir a linguagem Assembler.

### Detalhes da atividade

O processo de tradução e execução de uma linguagem de alto nível começa com um programa escrito em uma linguagem de alto nível sendo compilado para programa em assembly, e após essa operação ele é montado, através de um montador, em um módulo objeto em linguagem de máquina. O próximo passo é executado pelo link-editor que combina vários módulos com as rotinas de biblioteca para resolver todas as referências. Para finalizar, o Loader é encarregado de colocar o código de máquina nos locais apropriados da memória, para ser executado pelo processador Figura 3.2.

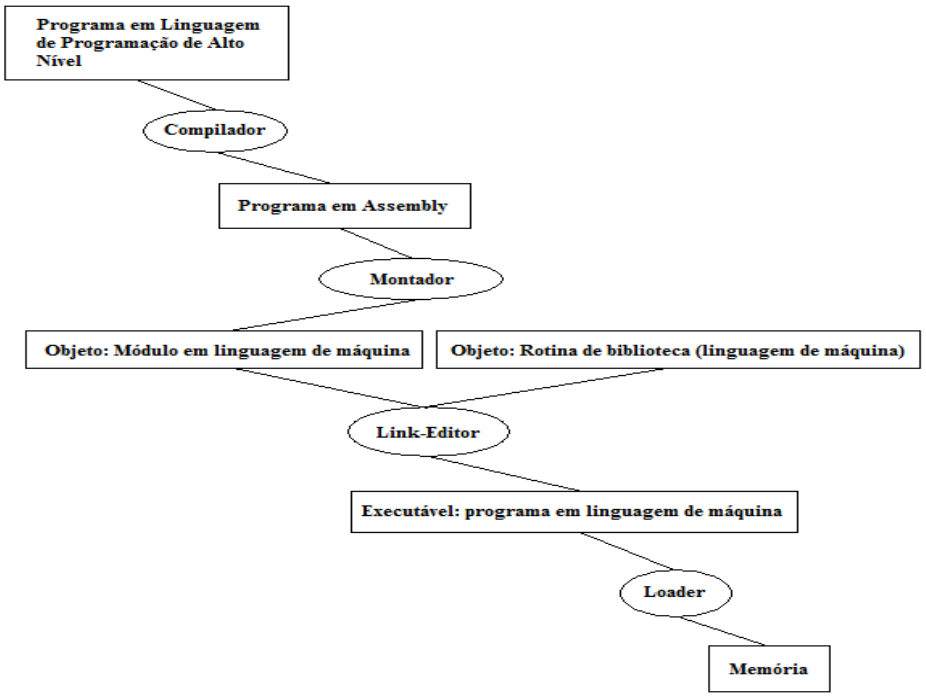



Figura 3.2: Etapas de Tradução de uma Linguagem de Programação

Fonte: http://www.devmedia.com.br/processo-de-traducao-e-execucao-de-programas/26872 consultado em 26-02-2016

Em alguns sistemas, algumas etapas são saltadas ou combinadas para agilizar o processo. As fases da compilação do de um programa assembler são ilustradas na figura 3.3.

Segundo Wikipédia, "Assembly" significa montagem. "Assembler" significa montador. Montagem é transformar uma sequência de código fonte (texto) em código objeto (linguagem de máquina), e montador é o programa que faz isso.

Na figura 3.3, o programador escreve um conjunto de comandos em modo texto, onde cada linha realiza funções específicas.

No entanto, o microcontrolador não entende estes comandos escritos em modo texto. Eles precisam ser traduzidos para uma linguagem binária (linguagem de máquina), e isso é feito pelo programa montador (assembler).

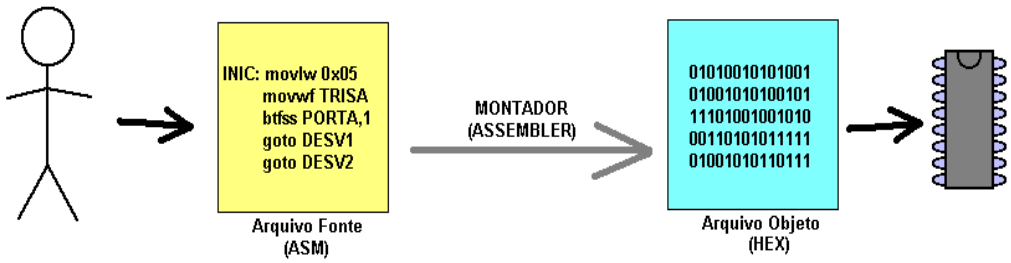



Figura 3.3: Compilação de um programa assembler

**Fonte: Adaptado da Wikipédia**

Programar diretamente em linguagem máquina é possível, mas é muito, mas muito mais difícil do que programar em assembly.

Utilizando a linguagem assembly e um programa montador, o programador não escreve em linguagem de máquina, mas sim uma linguagem textual, facilitando a construção dos programas. Embora fique mais fácil do que programar direto em linguagem de máquina, programar em Asembly ainda é uma das formas mais "difíceis" de programar, sendo conhecida como a linguagem de programação de "mais baixo nível".

O ficheiro fonte da Figura 3.3 (aquela lista de comandos digitada pelo programador) é composta de instruções (mnemônicos), parâmetros, rótulos, comentários e diretivas, e para poderem ser entendidos são transformados em linguagem de máquina por um programa montador.

Vamos refletir sobre cada um destes componentes do programa.

Instrução: É o nome dado a uma operação que o microcontrolador pode realizar. Por exemplo, se o microcontrolador pode realizar a soma de dois valores, dizemos que existe no mínimo uma instrução para soma.

Mnemônico: É uma representação textual de uma instrução. As instruções são, na verdade, códigos binários, e para serem entendidos pelos programadores devem ser representados na forma textual. Se os mnemônicos não fossem utilizados, teríamos que programar assembly utilizando códigos numéricos difíceis de memorizar.

Parâmetros: São as informações manipuladas por uma instrução. É necessário sempre que precisarmos informar à instrução quais os elementos envolvidos na operação, conforme a figura 3.4.


Figura 3.4: Parâmetros e instruções

**Fonte: Adaptado da Wikipédia**

Diretivas: São linhas que determinam como o programa montador irá trabalhar.
Não criam efeito direto no código binário produzido. Por exemplo, a diretiva LIST p=16F877 determina qual o microcontrolador que será utilizado.

Rótulos: São nomes dados às linhas do programa, e servem para que em uma instrução de desvio possamos determinar o ponto para onde se deseja ir no programa. Os rótulos sempre são alinhados na coluna 0 (sem espaços antes do mesmo), enquanto as instruções devem ser escritas após uma margem (obrigatoriamente após a coluna 0).

Comentários: São trechos de texto escritos após um sinal de ponto e vírgula (;). São úteis para que possamos adicionar pequenos lembretes no programa, facilitando a manutenção futura. Não interferem no tamanho do programa binário criado.

**Outros termos importantes:**

Montador: Transforma um programa fonte assembly em um programa executável. Um exemplo é o MPASM, que faz parte do MPLAB, uma ferramenta de desenvolvimento distribuída pela MicroChip (fabricante dos microcontroladores PIC).

**As instruções**

Por se tratar de um microcontrolador RISC, o PIC oferece um número reduzido de instruções. No entanto, ainda podemos dividir as instruções utilizadas pela família 16 dos microcontroladores PIC em 6 grupos:

- Instrução para manipulação de bytes de memória (B)

- Instrução para manipulação de bits de memória (b)
- Desvios incondicionais (Di)
- Desvios condicionais (D)
- Instruções com valores constantes (K)
- Instruções de controlo (G)

Tabela 3.1: instruções do PIC16F877

| Instrução e Parâmetros Mneumônicos | | Descrição | Tipo | Ciclos | Bits de status afetados |
|---|---|---|---|---|---|
| ADDWF | f, d | Adição : W + F. | B | 1 | C, DC, Z |
| ANDWF | f, d | E binário (AND) entre W e F, bit a bit. | B | 1 | Z |
| CLRF | f | Zera todos os bits de F. | B | 1 | Z |
| CLRW | | Zera todos os bits de W. | B | 1 | Z |
| COMF | f, d | Complemento de F. (bits com valores invertidos no byte) | B | 1 | Z |
| DECF | f, d | Decrementa F | B | 1 | Z |
| DECFSZ | f, d | Decrementa F e pula próxima linha se resultar zero | B,Dc | 1 (2) | |
| INCF | f, d | Incrementa F | B | 1 | Z |
| INCFSZ | f, d | Incrementa F e pula próxima linha se resultar zero | B,Dc | 1 (2) | |
| IORWF | f, d | OU inclusivo (OR) de W com F | B | 1 | Z |
| MOVF | f, d | Move F (geralmente utilizado para mover F para W) | B | 1 | Z |
| MOVWF | f | Move W para F | B | 1 | |
| NOP | | Operação nula. Nada é executado. | G | 1 | |
| RLF | f, d | Rotaciona F para esquerda com Carry Flag | B | 1 | C |
| RRF | f, d | Rotaciona F para direita com Carry Flag | B | 1 | C |

| | | | | | |
|---|---|---|---|---|---|
| SUBWF | f, d | Subtrai W de F : (f-W) | B | 1 | C,DC,Z |
| SWAPF | f, d | Troca os nibbles de f. Ex: (0xA3, após swap fica, 0x3A) | B | 1 | |
| XORWF | f, d | Ou exclusivo (XOR) entre W e F | B | 1 | Z |
| BCF | f, b | Apaga (clear) um bit de F | b | 1 | |
| BSF | f, b | Liga (set) um bit de F | b | 1 | |
| BTFSC | f, b | Testa um bit de F, pulando se for zero | b,Dc | 1 (2) | |
| BTFSS | f, b | Testa um bit de F, pulando se for um | b,Dc | 1 (2) | |
| ADDLW | k | Adiciona uma constante K em W | B | 1 | C,DC,Z |
| ANDLW | k | E (and) lógico de uma constante com W | B | 1 | Z |
| CALL | k | Faz uma chamada a uma subrotina | Di | 2 | |
| CLRWDT | | Limpa o Watchdog Timer (relógio do cão de guarda) | G | 1 | ~TO, ~PD |
| GOTO | k | Vá para. Um desvio para um outro ponto do programa. | Di | 2 | |
| IORLW | k | Ou inclusivo (OR) de uma constante com W | B | 1 | Z |
| MOVLW | k | Move uma constante para W | B | 1 | |
| RETFIE | | Retorna de uma interrupção | Di | 2 | |
| RETLW | k | Retorna de uma subrotina, movendo uma const. para W | B,Di | 2 | |
| RETURN | | Retorna de uma subrotina | Di | 2 | |
| SLEEP | | Vai para o modo standby | G | 1 | ~TO, ~PD |
| SUBLW | k | Subtrai uma constante de W | B | 1 | C,DC,Z |
| XORLW | k | Ou exclusivo (OR) entre W e uma constante | B | 1 | Z |

Fonte: www.microchip.com/PIC16F877A, consultado em 27-02-2016

**Como entender a tabela 3.1:**

1ª Coluna: instrução - Descreve todas as instruções utilizadas pelo microcontrolador PIC16F877. As instruções geralmente possuem nomes relacionados a suas funções.

2ª Coluna: parâmetros - Descreve os operandos utilizados pela instrução. Nesta coluna aparecem as letras f, d, b, k.

O (f) identifica que o parâmetro deve ser uma posição da memória RAM interna (que chamaremos de registradores). Os registradores serão explicados em breve e expressos em uma tabela.

O (d) identifica um parâmetro de destino, e pode valer W ou F. W é o registrador principal, e F é qualquer outro registrador.

O (b) é um parâmetro de identificação de um bit (0 a 7). Por exemplo, BSF PORTD,0 onde o (b) vale 0 ativa o bit menos significativo (bit 0) do registrador PORTB.

O (k) identifica que o parâmetro em questão é uma constante (rótulo ou valor fixo). Por exemplo, MOVLW 10 onde o valor de K é 10, move a constante 10 para o registrador principal.

3ª Coluna: descrição - Descreve a função dos operandos.

4ª Coluna – tipo - Define o grupo onde a instrução se encaixa. Veja a legenda no texto acima da tabela.

5º Coluna - ciclos - Uma instrução pode consumir 1 ou 2 ciclos de máquina. Cada ciclo de máquina, no caso dos microcontroladores PIC16F8xx, corresponde a 4 pulsos de clock. Portanto, se o cristal utilizado no microcontrolador for de 4MHz, ocorrerão 1MegaCiclos por segundo, ou seja, 1 milhão de ciclos por segundo (também usa-se 1mips - 1 milhão de instruções por segundo). Algumas instruções, portanto, demorarão 1/1000000 de segundos (1 microsegundo) para serem executadas, e outras demorarão 2/1000000 segundos (2 microsegundos). Algumas instruções (como os desvios condicionais) podem demorar 1 ou 2 ciclos, dependendo da condição avaliada pela instrução.

6ª Coluna - bits de status afetados - Inicialmente, devemos entender o que são BITS DE STATUS. De uma forma resumida, são "indicadores" existentes na memória do microcontrolador que registram informações sobre as operações realizadas (Exemplo: se a última operação resultou em zero ou não, se houve estouro no valor computado, etc...). Esta coluna visa descrever quais destes BITS DE STATUS são afetados pela instrução. Para saber mais sobre estes bits de status, procure bibliografia complementar.

**Registrador**

É o nome utilizado para identificar uma posição de memória interna do microcontrolador. No caso do microcontrolador PIC16F877, possuímos capacidade de acesso interno a 512 bytes de memória. Cada byte (8 bits) é um registrador.

Temos, portanto, 512 registradores. No entanto, alguns destes registradores são utilizados para fins específicos, estando diretamente relacionados a periféricos internos do microcontrolador (como conversores AD, entradas e saídas digitais, configuração dos periféricos, etc...), e outros destes registradores não são fisicamente implementados (são "buracos" deixados para futuras melhorias do projeto do microcontrolador). Outros registradores são os denominados "General Purpose Registers", ou "Registradores de utilização Geral". São posições de memória livres, que podem ser utilizados para armazenamento temporário de valores (variáveis). Como são parte da memória RAM do microcontrolador, todos os dados armazenados nos registradores são voláteis, ou seja, são perdidos ao se desligar o mesmo.

Listamos na tabela 3.2 os ficheiros registradores do PIC16F877.

**FIGURE 2-3: PIC16F877/876 REGISTER FILE MAP**

| Bank 0 | File Address | Bank 1 | File Address | Bank 2 | File Address | Bank 3 | File Address |
|---|---|---|---|---|---|---|---|
| Indirect addr.(*) | 00h | Indirect addr.(*) | 80h | Indirect addr.(*) | 100h | Indirect addr.(*) | 180h |
| TMR0 | 01h | OPTION_REG | 81h | TMR0 | 101h | OPTION_REG | 181h |
| PCL | 02h | PCL | 82h | PCL | 102h | PCL | 182h |
| STATUS | 03h | STATUS | 83h | STATUS | 103h | STATUS | 183h |
| FSR | 04h | FSR | 84h | FSR | 104h | FSR | 184h |
| PORTA | 05h | TRISA | 85h | | 105h | | 185h |
| PORTB | 06h | TRISB | 86h | PORTB | 106h | TRISB | 186h |
| PORTC | 07h | TRISC | 87h | | 107h | | 187h |
| PORTD(1) | 08h | TRISD(1) | 88h | | 108h | | 188h |
| PORTE(1) | 09h | TRISE(1) | 89h | | 109h | | 189h |
| PCLATH | 0Ah | PCLATH | 8Ah | PCLATH | 10Ah | PCLATH | 18Ah |
| INTCON | 0Bh | INTCON | 8Bh | INTCON | 10Bh | INTCON | 18Bh |
| PIR1 | 0Ch | PIE1 | 8Ch | EEDATA | 10Ch | EECON1 | 18Ch |
| PIR2 | 0Dh | PIE2 | 8Dh | EEADR | 10Dh | EECON2 | 18Dh |
| TMR1L | 0Eh | PCON | 8Eh | EEDATH | 10Eh | Reserved(2) | 18Eh |
| TMR1H | 0Fh | | 8Fh | EEADRH | 10Fh | Reserved(2) | 18Fh |
| T1CON | 10h | | 90h | General Purpose Register 16 Bytes | 110h | General Purpose Register 16 Bytes | 190h |
| TMR2 | 11h | SSPCON2 | 91h | | 111h | | 191h |
| T2CON | 12h | PR2 | 92h | | 112h | | 192h |
| SSPBUF | 13h | SSPADD | 93h | | 113h | | 193h |
| SSPCON | 14h | SSPSTAT | 94h | | 114h | | 194h |
| CCPR1L | 15h | | 95h | | 115h | | 195h |
| CCPR1H | 16h | | 96h | | 116h | | 196h |
| CCP1CON | 17h | | 97h | | 117h | | 197h |
| RCSTA | 18h | TXSTA | 98h | | 118h | | 198h |
| TXREG | 19h | SPBRG | 99h | | 119h | | 199h |
| RCREG | 1Ah | | 9Ah | | 11Ah | | 19Ah |
| CCPR2L | 1Bh | | 9Bh | | 11Bh | | 19Bh |
| CCPR2H | 1Ch | | 9Ch | | 11Ch | | 19Ch |
| CCP2CON | 1Dh | | 9Dh | | 11Dh | | 19Dh |
| ADRESH | 1Eh | ADRESL | 9Eh | | 11Eh | | 19Eh |
| ADCON0 | 1Fh | ADCON1 | 9Fh | | 11Fh | | 19Fh |
| General Purpose Register 96 Bytes | 20h | General Purpose Register 80 Bytes | A0h | General Purpose Register 80 Bytes | 120h | General Purpose Register 80 Bytes | 1A0h |
| | 7Fh | accesses 70h-7Fh | EFh F0h FFh | accesses 70h-7Fh | 16Fh 170h 17Fh | accesses 70h - 7Fh | 1EFh 1F0h 1FFh |

Unimplemented data memory locations, read as '0'.
* Not a physical register.

**Note 1:** These registers are not implemented on the PIC16F876.
**2:** These registers are reserved, maintain these registers clear.

Fonte: Tabela retirada do datasheet do microcontrolador PIC16F877

Após a leitura do texto disponibilizado para a realização desta atividade responda às questões abaixo:

Observando o trecho de programa abaixo, e tendo em mãos a Tabela 3.1 o que cada linha significa?

| | |
|---|---|
| 1 | volta |
| 2 | btfss PORTA,1 |
| 3 | goto deslig |
| 4 | ligado |
| 5 | movlw 0x0F |
| 6 | movwf PORTD |
| 7 | goto volta |
| 8 | deslig |
| 9 | goto volta |
| 10 | movlw 0xAA |
| 11 | movwf PORTD |
| 12 | goto volta |

## Conclusão

Nesta atividade vimos as fases da execução de um programa um escrito na linguagem de alto nivel e outro em assembler. Vimos também as vantagens e desvantagens de se programar a baixo nível e concluímos que a principal vantagem de programar em baixo nível é poder otimizar o código de modo a aproveitar ao máximo as características particulares do hardware.

A principal desvantagem é a grande desproporção que existe entre a complexidade do seu conjunto de instruções. Evimos também que um programa em assembly é composta por: Instrução, Mnemônico, Parâmetros, Diretivas, Rótulos, Comentários e Montador.

**Avaliação**

Este conteúdo será avaliado na avaliação sumativa da Unidade 3 e tem o peso de 5%.

## RESUMO DA UNIDADE

A execução de uma linguagem de alto nível começa com um programa escrito em alto nível que deve ser compilado para assembly, e de seguida ser montado, através de um montador, em um módulo objeto em linguagem de máquina. O próximo passo é executado pelo link-editor que combina vários módulos com as rotinas de biblioteca para resolver todas as referências. Para finalizar, o Loader é encarregue de colocar o código de máquina nos locais apropriados da memória.

A Compilação de um programa assembler processa-se em muito menos passos.

A principal vantagem de programar em linguagem de baixo nível é poder otimizar o código de modo a aproveitar ao máximo as características particulares do hardware.

A principal desvantagem de uma linguagem de baixo nível é a grande desproporção que existe entre a complexidade do seu conjunto de instruções.

## Avaliação da Unidade

Resolva o exercício utilizando o PIC

Para cada um dos exercícios abaixo, faça um novo programa e comente todas as linhas.

Utilização de GOTO

Escrever no bit B1 o que está no bit C1

Escrever no Bit B4 o que está no bit B0

Escrever no bit C3 o que está no bit A2

Escrever no bit A4 o que está no bit A5

Coloque na saída B0 o resultado da operação lógica AND entre A2 e B5

Coloque na saída B1 o resultado da operação lógica OR entre A2 e B5

Coloque na saída B2 o resultado da operação lógica XOR entre A2 e B5

Coloque na saída B3 e B4 o resultado da operação lógica NOT A2 , B5

Utilização de CALL

Converta os exercícios 5 a 8 para funções e utilize os bit C0 e C1 para escolher a operação pretendida.

Utilização de Páginas

Utilize o programa do exercício 9 e coloque cada uma das operações em páginas distintas da memória.

Contadores

Elabore um programa que simule um contador. O bit A0 é o clock, e a saída do contador é apresentada na porta C.

Acrescente ao programa anterior a possibilidade de fazer Reset através do bit A1 e utilize o bit A2 para indicar se a contagem é crescente ou decrescente.

Acrescente ao programa anterior a possibilidade de fazer Load com o auxílio do bit A3, que inicia a contagem com o que estiver presente na porta B.

Utilização de Timers

Utilize o programa anterior e em vez de utilizar o bit A0 para o clock, utilize o resultado do Timer0.

Utilização de Interrupt

Transforme o programa anterior utilizando a possibilidade de interrupt do Timer0.

Melhore o programa anterior de modo a garantir que o Load e Reset são assíncronos e minimize o tempo de permenância dentro da rotina de interrupt.

Utilização de valores Analógicos

Coloque na porta B o valor analógico da porta A0.

Coloque na porta C o valor analógico da diferença entre a porta A0 e A1.

Utilização de algoritmos

Faça um programa que realize a operação de multiplicação. O valor presente na porta A multiplicado pela B é colocado na porta C.

Faça um programa que realize a operação da divisão inteira. O valor presente na porta B dividido pelo A é colocado na porta C.

Utilizando o programa anterior, apresente na porta C o resto da mesma divisão inteira.

**Instruções**

As questões deverão ser respondidas individualmente.

Todas as questões devem ser respondidas em um ficheiro texto a ser enviado ao (à) instrutor(a) da disciplina através do e-mail.

**Critérios de Avaliação**

Esta avaliação tem o peso de 4% da nota final.

**LABORATÓRIO**

Arquitetura e Organização Avançada de Computadores:

**Objetivos do laboratório**

Criar programas utilizando linguem de baixo nível;

Criar programas utilizando linguem de Alto nível;

Exercitar a tradução da linguagem de baixo nível para alto e vice-versa

Detalhes do exercicio/atividades do laboratório

Objectivos do exercicios

Resolução de exercícios sobre a comparação linguagem de alto nível e baixo nivel

Recursos Necessarios

Computador, compiladores

Tempo Necessário

Duas Horas

Discrição do exercicio/atividade do laboratório

> Esta ficha está dividida em duas partes: a primeira parte sobre os ciclos de repetição, e a segunda sobre transferência de controlo.
>
> Cada uma das partes é composta por um primeiro exercício já resolvido.

**Laboratório**

**Parte 1**

Considere a seguinte função em C:

```
1  int dw_loop(int x, int y, int n)
2  {
3   do {
```

```
4      x += n;
5      y *= n;
6      n--;
7     } while ((n > 0) & (y < n));
8    return x;
9   }
```

Mostre o código em assembly que seria criado pelo comando:

gcc –O2 –S file_name.c

Considerando que os argumentos x, y, e n, passados para a função, se encontram respectivamente à distância 8, 12 e 16 do endereço especificado em %ebp, preencha a tabela de utilização de registos.

Identifique a expressão de teste e o corpo da função (body-statement) no bloco do código C, e assinale as linhas de código no programa em assembly que lhe são correspondentes.

Acrescente comentários ao programa em assembly.

**Resolução do Exercício 1:**

O ficheiro em assembly gerado pelo dado comando será semelhante ao seguinte:

```
1 movl 8(%ebp),%esi
2 movl 12(%ebp),%ebx
3 movl 16(%ebp),%ecx
4 .p2align 4,,7
5 .L6:
6 imull %ecx,%ebx
7 addl %ecx,%esi
8 decl %ecx
9 testl %ecx,%ecx
10 setg %al
11 cmpl %ecx,%ebx
12 setl %dl
```

```
13 andl %edx,%eax

14 testb $1,%al

15 jne .L6
```

Utilização dos registos:

| Registo | Variável | Atribuição inicial |
|---|---|---|
| %esi | x | x |
| %ebx | y | y |
| %ecx | n | n |
| %al | "temp1" | (n > 0) |
| %dl | "temp2" | (y > n) |

**Corpo da função:**

linhas 4 a 6 no código C;

linhas 6 a 8 no código assembly.

**Expressão teste:**

linha 7 do código C;

linhas 9 a 14, e a condição de salto na linha 15, no código assembly.

**Código anotado:**

```
1 movl 8(%ebp),%esi   Coloca x em %esi

2 movl 12(%ebp),%ebx   Coloca y em %ebx

3 movl 16(%ebp),%ecx   Coloca n em %ecx

4 .p2align 4,,7   Optimiza o desempenho da cache

5 .L6:   ciclo:

6 imull %ecx,%ebx   y *= n

7 addl %ecx,%esi   x += n

8 decl %ecx   n--

9 testl %ecx,%ecx   Testa n

10 setg %al   Coloca o valor lógico de (n>0) em %al
```

```
11 cmpl %ecx,%ebx   Compara y:n
12 setl %dl   Coloca o valor lógico de (y<n) em %dl
13 andl %edx,%eax   ((n > 0) & (y < n))
14 testb $1,%al   Testa o bit menos significativo
15 jne .L6   Se != 0, vai para o ciclo
```

**A seguinte porção de código assembly:**

```
1  movl 8(%ebp),%ebx
2  movl 16(%ebp),%edx
3  xorl %eax,%eax
4  decl %edx
5  js .L4
6  movl %ebx,%ecx
7  imull 12(%ebp),%ecx
8  .p2align 4,,7
9  .L6:
10 addl %ecx,%eax
11 subl %ebx,%edx
12 jns .L6
13 .L4:
```

**Foi obtida pela compilação de código C que tinha a seguinte estrutura:**

```
1 int loop(int x, int y, int n)
2 {
3  int result = 0;
4  int i;
5  for (i = ____; i ____ ; i = ___ ) {
6   result += _____ ;
7  }
8 return result;
```

```
9 }
```

Complete o programa em C de modo a obter-se um programa equivalente ao obtido com o código assembly. De notar que o resultado da função é devolvido no registo %eax.

Identifique a expressão de teste e o corpo da função (body-statement) no bloco do código C, e assinale as linhas de código no programa em assembly que lhe são correspondentes.

**Preencha a tabela de utilização de registos.**

Acrescente comentários ao programa em assembly.

**Parte 2**

Dada a seguinte função em C

```
1 int proc(void)
2 {
3 int x,y;
4 scanf("%x %x", &y, &x);
5 return x-y;
6 }
```

o gcc gera o seguinte código em assembly:

```
1 proc:
2 pushl %ebp
3 movl %esp,%ebp
4 subl $24,%esp
5 addl $-4,%esp
6 leal -4(%ebp),%eax
7 pushl %eax
8 leal -8(%ebp),%eax
9 pushl %eax
10 pushl $.LC0 Apontador para a string "%x %x"
11 call scanf
12 movl -8(%ebp),%eax
```

```
13 movl -4(%ebp),%edx

14 subl %edx,%eax

15 movl %ebp,%esp

16 popl %ebp

17 ret
```

Considere que a função proc começa a sua execução com os seguintes valores nos registos:

| Registo | Valor |
|---|---|
| %esp | 0x800040 |
| %ebp | 0x800060 |

Considere que a função proc invoca scanf (linha 11), e que scanf lê os valores 0x46 e 0x53 da standard input. Considere ainda que a string "%x %x" se encontra armazenada na memória em 0x300070.

Que valor é colocado no registo %ebp na linha 3?

Em que endereços estão as variáveis locais x e y armazenadas?

Qual é o valor de %esp na linha 11?

Desenhe um diagrama da stack frame para proc logo após o regresso de scanf. Inclua o máximo de informação que possua sobre endereços e conteúdos dos elementos da stack frame.

Indique as regiões da stack frame que não são usadas por proc (estas áreas desperdiçadas são usadas para melhorar o desempenho da cache).

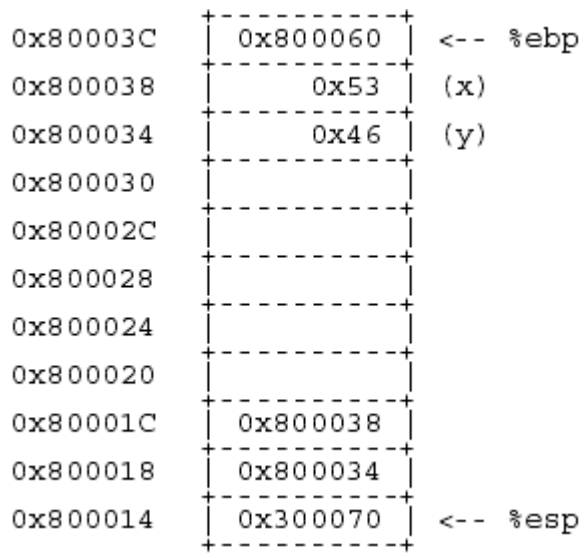
**Resolução do Exercício 3:**

Começa-se com o valor 0x800040 em %esp. A linha 2 decrementa-o por 4, dando 0x80003C, e este passa a ser o novo valor de %ebp.

Podemos ver como é que as 2 instruções leal calculam o valor dos argumentos a passar a scanf. Uma vez que os argumentos são "empurrados" para a stack pela ordem inversa, podemos ver que x se encontra deslocado de –4 células relativo a %ebp e que y está com um deslocamento de –8. Os endereços de memória são então 0x800038 e 0x800034.

Começando com o valor original de 0x800040, a linha 2 decrementou o valor de SP de 4 unidades, a linha 4 de 24, e a linha 5 de 4, e as 3 instruções de push decrementaram de 12,; no total houve uma variação de 44. Assim, na linha 11 %esp tem o valor 0x800014.

A stack frame tem a seguinte estrutura e conteúdo:




As células com os endereços 0x800020 a 0x800033 não são usadas.

Considere a mesma função proc, do exercício anterior, mas que agora a função proc começa a sua execução com os seguintes valores nos registos:

| Registo | Valor |
|---|---|
| %esp | 0x800060 |
| %ebp | 0x800080 |

Também scanf lê agora os valores 0x10 e 0x35 da standard input. Considere ainda que a string "%x %x" se encontra armazenada na memória em 0x400240.

Qual o valor do apontador para a stack frame desta função?

Qual o registo correspondente ao apontador para a stack frame?

Em que endereços estão as variáveis locais x e y armazenadas?

Em que endereço é armazenado o endereço de retorno da função scanf?

O endereço de retorno da função scanf corresponde a que registo?

Desenhe um diagrama da stack frame para proc logo após o regresso de scanf. Inclua o máximo de informação que possua sobre endereços e conteúdos dos elementos da stack frame.

A seguinte sequência de código aparece quase no início do código assembly gerado por GCC para uma função C:

```
1 pushl %edi
2 pushl %esi
3 pushl %ebx
4 movl 24(%ebp),%eax
5 imull 16(%ebp),%eax
6 movl 24(%ebp),%ebx
7 leal 0(,%eax,4),%ecx
8 addl 8(%ebp),%ecx
9 movl %ebx,%edx
```

Pode-se ver que 3 registos (%edi, %esi, e %ebx) são salvaguardados na stack. O programa altera no seu corpo não apenas estes, mas outros 3 registos (%eax, %ecx, e %edx). No fim da função, o valor dos registos %edi, %esi, e %ebx são recuperados usando a instrução popl, enquanto os outros 3 são deixados com os seus valores entretanto modificados.

Explique esta aparente inconsistência na salvaguarda e recuperação de registos na codificação de funções.

Que nome tem cada um dos tipos de registos apresentados.

**Resultados e requisitos de apresentação**

As atividades devem ser respondidas em um documento de texto enviadas para o correio eletrónico do(a) instrutor(a). Quem as resolveu fazendo a utilização de lápis, ou caneta, deverá fazer o scanner do conteúdo e enviar ao instrutor.

**Critérios de avaliação**

Este laboratório tem um peso de 1%

# Unidade 4: Interface Entrada/ Saida

## Introdução à Unidade

Este Unidade examina o sub-sistema de entrada e saída (E/S). Neste sub-sistema estão incluídas as interfaces de E/S, através das quais os dispositivos periféricos são conetados ao sistema.

Este capítulo inicia descrevendo como processador e interfaces de e/s se comunicam, a organização típica de uma interface de e/s, e como o processador exerce controlo sobre um dispositivo periférico através de uma interface de e/s. Em seguida são apresentadas as principais técnicas de transferência de dados em operações de e/s. Por último, são discutidos alguns aspectos relacionados com barramentos de e/s.

## Objetivos da Unidade

Após a conclusão desta unidade, você deverá ser capaz de:

- Identificar e descrever as formas de Interação entre Processador e Interfaces de E/S;
- Reconhecer, na organização típica de um computador, o sub-sistema de E/S
- Identificar as partes que constituem a organização de uma Interface de E/S;
- Identificar técnicas de Transferência de Dados;
- Distinguir e caracterizar Padrões de Barramento.

**TERMOS-CHAVE**

**Instruções:** é uma operação única executada por um processador e definida por um conjunto de instruções. Num sentido amplo, uma "instrução" pode ser qualquer representação de um elemento num programa executável, tal como um bytecode.

**CISC:** Conjunto Complexo de Instruções - é uma linha de arquitetura de processadores capaz de executar centenas de instruções complexas diferentes sendo, assim, extremamente versátil.

**RISC:** Conjunto Reduzido de Instruções - linha de arquitetura de processadores que favorece um conjunto simples e pequeno de instruções que levam aproximadamente à mesma quantidade de tempo para serem executadas.

**Atividades de Aprendizagem**

## Atividade 1 – Interação entre Processador e Interfaces de E/S

**Introdução**

Em sistemas tais como microcomputadores e estações de trabalho, as interfaces de E/S são ligadas ao processador através de barramentos de endereço, dados e controle, de maneira semelhante à conexão entre memória principal e processador.

Nesta atividade vamos refletir e exercitar sobre a Interação entre Processador e Interfaces de Entrada/Saída (E/S), vamos identificar as formas de Interação entre Processador e Interfaces de E/S, reconhecer a organização de uma Interface de E/S, identificar técnicas de transferência de Dados e Aprender Padrões de Barramentos.

**Detalhes da atividade**

A organização típica de um computador incluindo o sub-sistema de E/S é mostrada na Figura 4.1.

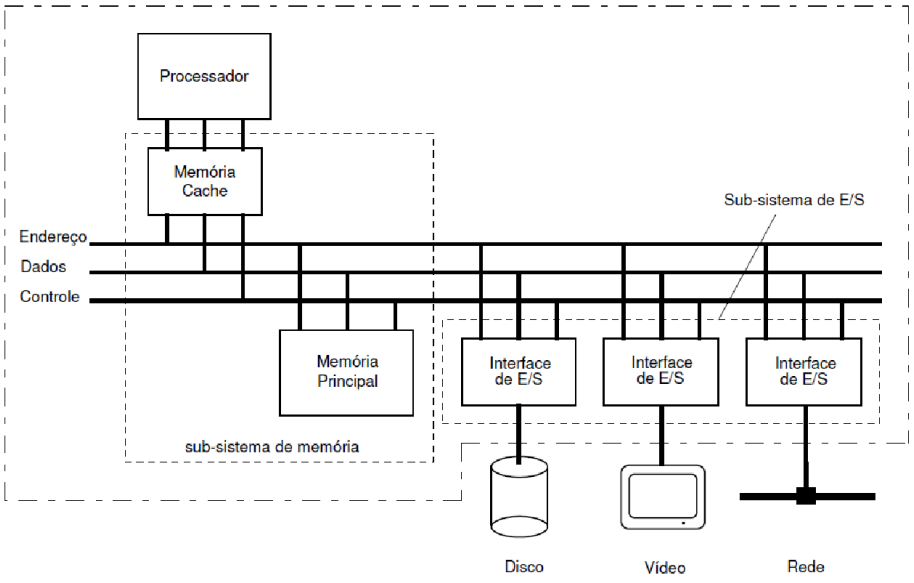



Figura 4.1: Arquitetura de um computador, incluindo o sub-sistema de e/s.

**Fonte: Adaptado de TANENBAUM, 2007**

Assim como acontece em relação à memória principal, o processador realiza acessos de leitura ou de escrita a uma interface de E/S. Em um acesso de leitura, o processador obtém um dado recebido do dispositivo periférico conetado à interface, ou então uma informação de estado sobre uma operação de E/S em andamento ou recém-completada.

Em um acesso de escrita, o processador fornece à interface um dado que deve ser enviado ao dispositivo periférico, ou então o código de um comando que inicia uma operação de e/s ou uma operação de controlo sobre o dispositivo periférico.

Cada interface de E/S é identificada por um endereço único. Em um acesso de leitura, o processador coloca o endereço da interface no barramento de endereço e ativa um sinal de leitura. Após um certo intervalo de tempo, a interface coloca a informação desejada no barramento de dados. O processador termina o ciclo de barramento a ler a informação presente no barramento de dados e a retirar o endereço e o sinal de controlo.

Em um acesso de escrita, o processador coloca o endereço da interface e o dado nos respectivos barramentos, e ativa um sinal de escrita. A interface selecionada armazena a informação presente no barramento de dados. No final do ciclo de barramento, o processador retira o endereço e o dado e desativa o sinal de controlo. Assim como nos ciclos de barramento com a memória, todos estes eventos são comandados pelo processador e ocorrem em sincronismo com o sinal de clock.

Após leitura atenta dos detalhes da atividade 4.1 faça um resumo sobre a Interação entre Processador e Interfaces de E/S e envie em forma de apresentação digital para o(a) instrutor(a) da disciplina através do correio eletrónico.

## Conclusão

Nesta atividade refletimos sobre a Interação entre Processador e Interfaces de Entrada/Saída e concluímos que as interfaces de E/S são ligadas ao processador através de barramentos de endereço, dados e controle. O processador realiza acessos de leitura ou de escrita a uma interface de E/S.

Concluimos ainda que cada interface de E/S é identificada por um endereço único.

**Avaliação**

Este conteúdo será avaliado na avaliação sumativa da Unidade 4 e tem o peso de 5%.

## Atividade 2 - Organização de uma Interface de E/S

**Introdução**

A principal função de uma interface de E/S é tornar transparente para o processador os detalhes de operação e controle dos dispositivos periféricos.

Nesta atividade vamos compreender a organização de uma interface de E/S.

**Detalhes da atividade**

Podemos considerar que uma interface de E/S está organizada em duas partes, como mostra a Figura 4.2.

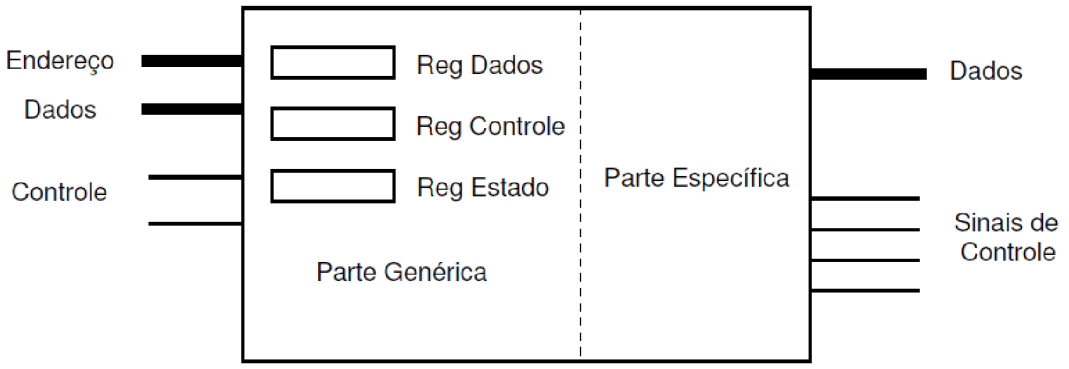



Figura 4.2: Organização típica de uma interface de E/S.

**Fonte: Adaptado de TANENBAUM, 2007.**

A parte genérica, como o próprio nome indica, é semelhante entre os diferentes tipos de interfaces de E/S. É esta porção da interface que é vista pelo processador. Em geral, na parte genérica existem alguns registradores, cujo número e função depende em parte do tipo de periférico acoplado à interface. No entanto, como mostra a figura 4.2, na maioria das interfaces a parte genérica inclui pelo menos um registrador de dados, um registrador de controlo e um registrador de estado. O acesso a cada um destes registradores é feito pelo processador através de um endereço de E/S diferente.

O registrador de dados é utilizado para as transferências de dados entre o processador e o dispositivo periférico. Em uma operação de saída, o processador escreve um dado neste registrador e a interface se encarrega de enviá-lo para o periférico. No sentido contrário, em uma operação de entrada, a interface recebe um dado do periférico e o armazenamento no registrador de dados. O processador executa então um acesso de leitura à interface e obtém o dado depositado no registrador.

O processador utiliza o registrador de controlo para enviar comandos à interface. Este comando é enviado sob a forma de um código. Cada interface possui um repertório de comandos próprio. Quando o processador escreve um comando no registrador de controlo, a interface interpreta o código do comando e executa a operação solicitada, que pode ser uma operação interna à interface ou sobre o periférico a ela conectado. Finalmente, o registrador de estado é utilizado para veicular informações sobre uma operação de E/S. Tipicamente, este registrador possui bits para indicar o término de uma operação e para indicar condições de erro que eventualmente possam acontecer durante a operação.

A parte específica interage diretamente com o periférico, e por isso ela difere bastante entre os diferentes tipos de interfaces.

No entanto, apesar das diferenças, a parte específica na maioria das interfaces possui dois conjuntos de sinais. Um deles é a própria via através da qual são transferidos os dados entre a interface e o periférico. O outro conjunto é formado pelos sinais utilizados no controle do periférico.

Como exemplo de interface de E/S, a Figura 4.3 mostra a organização simplificada de uma interface para unidades de disco rígido, a interface Intel 82064. Em sua parte genérica, esta interface possui sete registradores. O data register registrador é utilizado na transferência de dados. O command register equivale ao registrador de controlo descrito anteriormente. Algumas operações exigem informações adicionais, que são escritas pelo processador nos registradores de parâmetro. O cylinder register é o registrador de parâmetro onde o processador escreve o número do cilindro (trilha) onde será feito o acesso. Os registradores sector number register e sector count register servem para indicar, respectivamente, o número do setor inicial e a quantidade de setores que devem ser acessados a partir do setor inicial. No sector/drive/head register o processador escreve o tamanho do setor em bytes, o número da unidade de disco e o número da cabeça.

Finalmente, o error register indica alguma condição de erro ocorrida.

Na parte específica, a interface possui dois circuitos que realizam a leitura e a escrita de dados no disco. Um terceiro circuito controla a parte mecânica da unidade de disco. Por exemplo, este circuito gera os sinais STEP, que fazem a cabeça avançar, e STEP DIR, que indica a direção de avanço da cabeça. Este circuito também recebe sinais, como o sinal READY, que indicam o estado da unidade de disco.

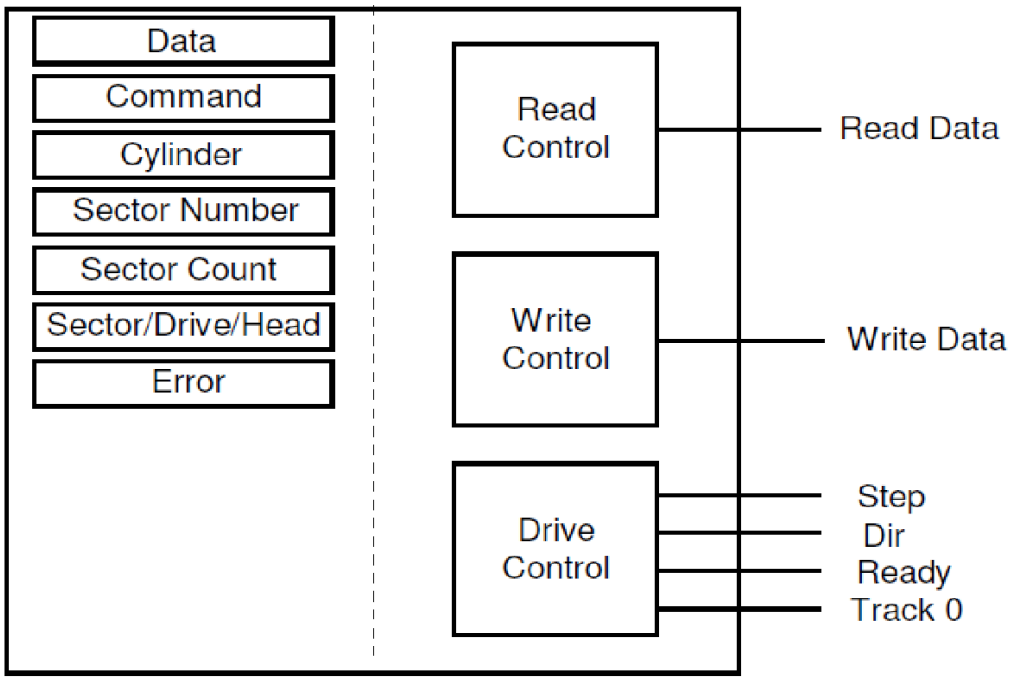



Figura 4.3: Organização típica de uma interface de E/S.

**Fonte: Adaptado de TANENBAUM, 2007**

Este exemplo mostra a organização típica de uma interface de E/S. De um lado, a interface possui registradores através dos quais o processador envia e recebe dados, indica o tipo e os parâmetros da operação de E/S e obtém informações sobre o sucesso da operação. Do outro lado, a interface possui os circuitos e sinais necessários para controlar um particular periférico. Organização semelhante pode ser encontrada em interfaces para vídeo, impressoras, redes locais, entre outros. Para auxiliar a compreensão integral remendamos a agruparem em grupo de dois alunos e fazerem uma pesquisa na internet sobre Organização de uma Interface de E/S. O relatório do trabalha deverá ser enviado em forma de apresentação digital para o instrutor da disciplina através do correio eletrónico.

## Conclusão

Nesta atividade debruçamos sobre a organização de uma interface de E/S e concluímos que ela é organizada em duas partes: A parte genérica que é semelhante entre os diferentes tipos de interfaces de E/S e a parte específica: interage diretamente com o periférico.

**Avaliação**

Este conteúdo será avaliado na avaliação sumativa da Unidade 4 e tem o peso de 5%.

## Atividade 3 – Padrões de Barramentos

**Introdução**

A Figura 4.4 mostra a arquitetura de sistema típica de microcomputadores e algumas estações de trabalho, onde processador, memória principal e interfaces de E/S estão interligados através de um conjunto de três barramentos. Estes barramentos são chamados coletivamente de barramento de sistema. Algumas características do barramento de sistema, tais como largura do barramento de endereço e do barramento de dados, são determinadas pelo processador. Outras características estão relacionadas com o sub-sistema de E/S, como por exemplo, o número de sinais de interrupção e o número de canais de DMA disponíveis no barramento de controlo.

**Detalhes da atividade**

Na categoria de microcomputadores, foram estabelecidos alguns padrões de barramento de sistema, com a finalidade de garantir a compatibilidade com interfaces de e/s de diferentes fabricantes. A figura 4.4 relaciona alguns padrões de barramentos de sistema em microcomputadores e suas principais características.

| Padrão | Endereço | Dados | Interrupções | Canais DMA | Desempenho (E/S, DMA) |
|---|---|---|---|---|---|
| IBM PC XT | 20 bits | 8 bits | 8 | 4 | 62.5 Kbytes/s |
| ISA | 24 bits | 16 bits | 15 | 7 | 100 Kbytes/s |
| EISA | 32 bits | 32 bits | ilimitado | 7 | 33 Mbytes/s |
| PCI | 32 bits | 32/64 bits | ilimitado | | 132/264 Mbytes/s |

Figura 4.4: Características de alguns padrões de barramentos de sistema

**Fonte: Adaptado de TANENBAUM, 2007**

O barramento de sistema no IBM PC XT era, na realidade, uma simples extensão dos barramentos do processador Intel 8088. O número de sinais de interrupção e de canais de DMA era bastante limitado. O IBM PC XT foi substituído pelo IBM PC AT, cujo barramento de sistema tornou-se o padrão ISA (Industry Standard Architecture). Com o ISA, o número de sinais de interrupção e de canais de DMA foi quase duplicado, enquanto o desempenho das operações de E/S com DMA aumentou em 60%. Apesar da diferença na largura dos barramentos de endereço e de dados, o padrão ISA permite a conexão de interfaces originalmente projetadas para o barramento IBM PC XT.

Com o lançamento dos processadores Intel 80386 e 80486, aumentaram as exigências quanto ao desempenho de E/S. Para fazer face à estas exigências, foi criado o padrão ISA estendido, ou EISA (Extended Industry Standard Architecture).

No padrão EISA, transferências de dados em operações de E/S são controladas por um componente especial, denominado bus master. Na verdade, um bus master está contido em uma interface de e/s sofisticada, que normalmente possui uma lógica dedicada ao controle do barramento e por uma memória local. Em um barramento EISA podem existir até 15 bus masters.

O bus master executa ciclos de barramento para transferir dados entre ele mesmo e a memória principal, ou ainda entre ele e uma interface de e/s. Como as transferências são sempre controladas pelo bus master, a memória ou a interface de e/s são denominados slaves. O bus master pode realizar dois tipos de transferências. No primeiro tipo, chamado standard transfer, o bus master e o slave executam um protocolo de sincronização a cada dado transferido. No modo burst transfer, este protocolo de sincronização é executado apenas uma vez, no início da transferência de um bloco de dados. Assim, transferências no modo burst transfer são mais rápidas que no modo standard transfer. Tipicamente, no modo standard transfer, são necessários 8 ciclos de clock para transferir um dado, enquanto no modo burst transfer a sincronização inicial consome 4 ciclos de clock e apenas 1 ciclo adicional para cada dado transferido.

O modo burst transfer foi introduzido no padrão EISA para atender às necessidades de periféricos que transferem blocos de dados com uma alta taxa de transferência.

O padrão EISA também permite transferências utilizando as tradicionais técnicas de interrupção e de acesso direto à memória, mas com vantagens. Nos barramentos PC XT e ISA, existe um número máximo de sinais para pedidos de interrupção disponíveis no barramento de controlo, o que limita o número de interfaces que podem usar a técnica de interrupção. No barramento EISA, não existe limitação quanto ao número de interfaces que podem solicitar interrupções.

**Além disso, transferências via DMA são mais rápidas.**

Um ponto importante no padrão EISA é a compatibilidade. Como o próprio nome sugere, o EISA foi criado como uma extensão de barramentos anteriores. Isto significa que interfaces para os barramentos PC XT e ISA podem ser utilizados em sistemas com barramento EISA.

Atualmente, os computadores são fabricados segundo o padrão PCI. Este padrão é uma interface de 64 bits num pacote de 32 bits. O barramento PCI roda em 33 MHz e pode transferir 32 bits de dados em cada ciclo de clock. Uma característica importante dos padrões de barramento mais recentes, como o EISA e o PCI, é o compartilhamento de interrupções. Esta técnica permite que dois dispositivos utilizem a mesma interrupção.

**Barramentos Locais**

Normalmente, um barramento de sistema possui conectores (os chamados slots) destinados à conexão das interfaces de e/s ao sistema. Em geral, o barramento de sistema inclui um número razoável de conectores, de forma a permitir eventuais expansões com o acréscimo de novas interfaces.

Devido ao grande número de conectores, o barramento torna-se fisicamente longo. Infelizmente, quanto maior o comprimento do barramento, maior será o efeito de alguns fatores elétricos indesejáveis que limita a taxa de transferência de dados. Esta limitação afeta não somente transferências de dados em operações de e/s, mas também os acessos à memória. Isto acontece porque um mesmo barramento é utilizado tanto para acessar a memória quanto para conectar as interfaces de e/s ao sistema. Para eliminar os efeitos negativos de um único barramento sobre os acessos à memória, alguns sistemas utilizam dois barramentos distintos, como mostra a Figura 4.5. Nesta organização, existe um barramento local que interliga o processador à memória principal e um barramento de e/s à parte, utilizado apenas para a conexão das interfaces de e/s. O barramento de e/s é isolado do barramento local através de um adaptador.

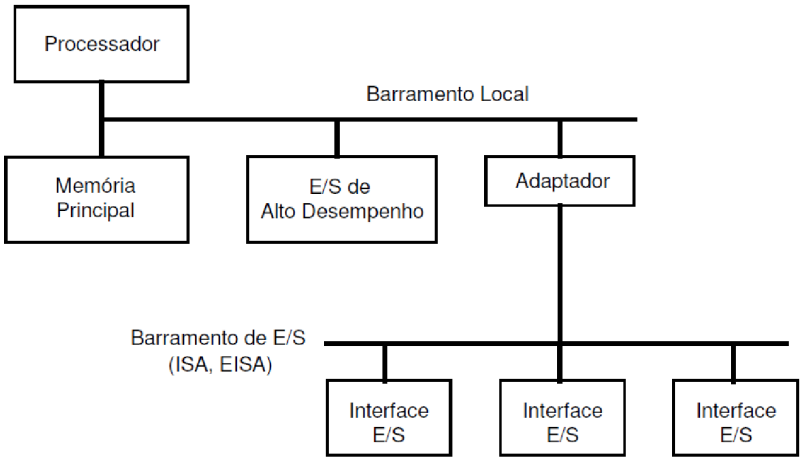



Figura 4.5: Estrutura de um sistema com dois barramentos.

**Fonte: Adaptado de TANENBAUM, 2007**

Com barramentos diferentes, a comunicação entre processador e memória principal fica isolada dos efeitos negativos decorrentes do comprimento excessivo do barramento de E/S. O comprimento do barramento local é extremamente reduzido: a este barramento podem ser conectados apenas algumas poucas (no máximo duas ou três) interfaces de e/s de alto desempenho, como por exemplo, interfaces de vídeo. Atualmente existem alguns padrões de barramentos locais estabelecidos. Os mais comuns são a VESA Local-Bus, desenvolvido pela Video Equipment Standards Association, e o Peripheral-Connect Interface, ou PCI, desenvolvido pela Intel Corp. Com o aumento do desempenho dos microprocessadores, existe uma tendência que arquiteturas com barramentos distintos venham a ser adotadas em uma faixa cada vez maior de sistemas.

**Barramentos de Periféricos**

Em alguns sistemas é possível encontrar um terceiro tipo de barramento, denominado barramento de periféricos. Este nível de barramento tem como principal objetivo prover um padrão de compatibilidade entre diferentes fabricantes de dispositivos periféricos. Em geral, o barramento de periférico é conectado ao barramento de E/S através de uma interface adaptadora especial.

Atualmente, o principal padrão de barramento de periféricos é o SCSI (Small Computer Systems Interface). A primeira versão do padrão SCSI foi lançada em 1986. A segunda versão, SCSI 2, foi lançada em 1990. O barramento SCSI possui uma estrutura em árvore, como mostra a Figura 4.6.

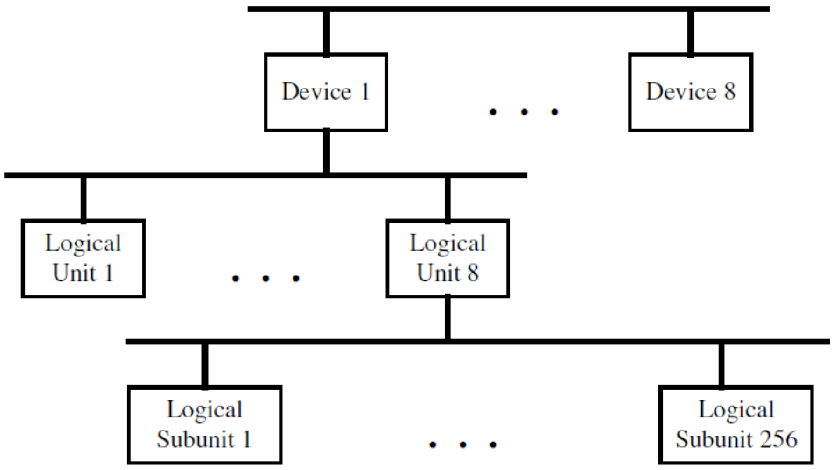



Figura 4.6: Estrutura do barramento de e/s no padrão SCSI.

**Fonte: Adaptado de TANENBAUM, 2007**

No primeiro nível podem existir até 8 SCSI dispositivos. Existem dois tipos de SCSI dispositivos: o primeiro tipo é chamado host adapter, e é utilizado para conectar o barramento SCSI ao barramento de e/s do sistema; o segundo tipo é chamado peripheral controller. Cada controller pode ser uma interface de e/s ou um adaptador para o próximo nível do barramento.

A cada controller podem estar ligados até 8 logical units no nível abaixo do barramento. Cada logical unit pode ser um periférico de e/s ou um adaptador para o próximo nível do barramento. Finalmente, a cada logical unit podem estar conectados até 256 logical subunits no terceito nível do barramento. Cada logical subunit corresponde a um periférico de e/s.

O padrão SCSI 2 apresenta dois modos especiais de operação. O primeiro modo é chamado fast SCSI. Enquanto a velocidade no modo básico de operação é de 4 milhões de transações por segundo, o modo fast SCSI permite uma velocidade de 10 milhões de transações por segundo. No entanto este modo exige um hardware especial, que aumenta o custo do barramento. O outro modo de operação é chamado wide SCSI. Neste modo são usadas 32 linhas de dados, ao invés das 16 linhas usadas no modo básico. Os modos fast SCSI e wide SCSI podem ser utilizados em conjunto, possibilitando uma taxa de transferência de 40 Mbytes/s.

Para auxiliar a compreensão integral dos conteúdos desta atividade, recomendamos a visualização dos videos:

"Aula 09 Barramentos". Disponível em:

https://www.youtube.com/watch?v=B5aSpqChOYQ consultado em 27-02-2015.

" Aula 08 Barramentos". Disponível em:

https://www.youtube.com/watch?v=oUIH9t_8kaE consultado em 27-02-2015.

Faça um pequeno comentário sobre o vídeo assistido e ao texto lido e envie em forma de apresentação digital para o (a) instrutor (a) da disciplina através do correio eletrónico.

## Conclusão

Na categoria de microcomputadores, foram estabelecidos alguns padrões de barramento de sistema, com a finalidade de garantir a compatibilidade com interfaces de E/S de diferentes fabricantes.

**Avaliação**

Este conteúdo será avaliado na avaliação sumativa da Unidade 4 e tem o peso de 5%.

**RESUMO DA UNIDADE**

Nesta unidade refletimos e exercitamos sobre a Interação entre Processador e Interfaces de Entrada/Saída (E/S), demonstramos a forma como se processa a Interação entre Processador e Interfaces de E/S, identificamos a organização de uma Interface de E/S, aprendemos a distinguir os Padrões de Barramentos.

## Avaliação da Unidade

Responda as seguintes questões abaixo:

Explique a função das interfaces de E/S.

Diferencie: a. Comunicação serial e paralela transmissão b. Síncrona e assíncrona. c. Transmissão simplex, half-duplex e full-duplex.

Indique o papel dos barramentos de: dados, endereço e controle.

Qual o objetivo da arbitragem de barramento? Cite os tipos de arbitragem possíveis e explique a diferença entre os mesmos.

O que é protocolo de barramento?

Defina barramento multiplexado e explique seu funcionamento.

Indique as principais características do barramento ISA (Industry Standard Architecture).

Indique as principais características do barramento padrão PCI (Peripheral Component Interconnect).

**Instruções**

As questões deverão ser respondidas individualmente.

Todas as questões devem ser respondidas em um ficheiro texto a ser enviado ao (à) instrutor (a) da disciplina através do correio eletrónico.

**Critérios de Avaliação**

Esta avaliação tem o peso de 5% da nota final.

## RESUMO DA UNIDADE

O módulo Arquitetura e Organização Avançadas de Computadores foi estruturado em cinco unidades:

Unidade 0: Introdução a Arquitetura e Organização Avançada de Computadores:

O propósito desta unidade foi verificar a compreensão e conhecimentos dos (as) estudantes sobre tópicos relacionados com este módulo. Rever alguns conceitos importantes.

Unidade 1: Organização Funcional do computador: Nesta unidade foram abordado os conceitos relacionados com a organização funcional de computadores. Nela, foram fornecidas algumas técnicas e conceitos básicos que nos ajudaram na compreensão e análise da interação entre hardware e software.

Unidade 2: Multi-processamento: Nesta Unidade, foram estudadas as noções do multiprocessamento por forma a permitir o entendimento de como um processador pode suportar a execução simultânea de programas.

Unidade 3: Programação em Baixo Nível: permitiu-nos Identificar as limitações de arquiteturas de baixo nivel, conhecer a estrutura de programas de baixo nível e aprender a arquitetura de suporte às linguagens de baixo e alto nível e linguagem assembler.

Unidade 4: Interface Entrada/Saida: permitiu examinar o sub-sistema de entrada e saída (e/s). Neste sub-sistema estão incluídas as interfaces de e/s, através das quais os dispositivos periféricos são conectados ao sistema.

**Avaliação do Curso**

O curso é avaliado através das avaliações sumativas disponibilizadas em cada uma das Unidades valendo 5% cada e totalizando 25% da nota final, Será igualmente aplicado um exame intercalar cujo peso será de 20% e um Exame final com o peso de 55%.

**Exame Intercalar**

Qual a diferença entre dispositivos mestres e escravos de um barramento?

Indique as transações executadas por um barramento assíncrono.

Liste as vantagens e desvantagens dos barramentos síncronos e assíncrono.

A porção de código seguinte, apresentado em assembly do IA32, resultou da compilação do respectivo código fonte escrito em linguagem C.

Apresente uma tabela que relacione os registos do processador usados com as variáveis declaradas na função escrita em C.

Analise o código assembly de forma a poder completar os espaços em falta da função escrita em C.

Indique que optimização produziu o compilador para esta função.

A seguinte função permite ler uma linha de caracteres do dispositivo standard de entrada e retornar o seu comprimento.

Com base no código assembly relacionado, apresente o diagrama do stack frame no momento em que a função fgets() é evocada.

Assinale de forma correspondente os elementos do stack frame que contêm os parâmetros a passar à função fgets().

O código que se segue em linguagem C gerou o seguinte código em assembly. No entanto se o compilador fosse mais "inteligente" poderia ter substituído as linhas assinaladas com um asterisco por uma única instrução assembly. Indique qual e como.

**Instruções**

As questões deverão ser respondidas individualmente.

Todas as questões devem ser respondidas em um ficheiro texto que deve ser enviado ao (à) instrutor(a) da disciplina através do correio electrónico.

**Critérios de Avaliação**

A avaliação intercalar tem o peso de 20% da nota final.

**Exame Final**

1. Linguagem de alto nível (HLL)

Se escrever o seguinte programa em C e o gravar como soma.c:

```
int res=0;
```

```
int soma (int x, int y)
{
  int t = x + y;
  res += t;
  return (t);
}
```

Em que formato se encontra a informação do código fonte, acima apresentado?

Que comando utilizaria para tentar criar um programa executável? Conseguiria criá-lo a partir do código acima apresentado? Porquê?

**Compilação**

Por compilação entende-se a conversão de um programa escrito num dado nível de abstracção noutro de nível inferior.

a) Que comando utilizaria para o gcc criar o código Assembly correspondente, apresentado a seguir:

```
  pushl   %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %eax
    addl    8(%ebp), %eax
    addl    %eax, res
    popl    %ebp
    ret
```

b) Em que formato se encontra a informação do código Assembly?

Este programa pode ser executado directamente pela máquina? Em que nível de abstracção nos encontramos?

**Montagem (Assembler)**

Se utilizar o comando:

```
    gcc –c soma.s
```

Qual é o conteúdo do ficheiro que será criado?

Em que formato se encontra a informação contida nesse ficheiro?

Com que aplicação, ou aplicações, se poderia visualizar essa informação?

Suponha que o código resultante era:

```
0x0 <soma>:      0x55    0x89    0xe5    0x8b    0x45    0x0c    0x03
0x45

0x8 <soma+8>:    0x08    0x01    0x05    0x00    0x00    0x00    0x00
0x5d

0x10 <soma+16>: 0xc3    0x8d    0x76    0x00
```

O que representam estes valores?

Este programa pode ser executado directamente pela máquina? Em que nível de abstração nos encontramos?

Suponha que agora visualizava o mesmo código, mas com a seguinte apresentação:

```
00000000 <soma>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   8b 45 0c                mov    0xc(%ebp),%eax
   6:   03 45 08                add    0x8(%ebp),%eax
   9:   01 05 00 00 00 00       add    %eax,0x0
   f:   5d                      pop    %ebp
  10:   c3                      ret
  11:   8d 76 00                lea    0x0(%esi),%esi
```

Quantas instruções têm a função soma? Quantos bytes ocupam?

Linker

a) Assumindo que tínhamos outro programa, cujo main chamava a função soma definida no programa soma.c, que comando usaria para criar um executável?

b) Em que formato se encontraria a informação desse programa executável?

**Critérios de avaliação**

O Exame final tem o peso de 55% da nota final.

## Referências do Curso


- Alves, Marco, Avaliação do Compartilhamento das Memórias Cache no Desempenho de Arquiteturas Multi-Core –
- http://www.lume.ufrgs.br/bitstream/handle/10183/16129/000697073.pdf?sequence=1 consultado em 4 de Janeiro de 2015.
- Breshears, C. The Art of Concurrency. San Francisco, CA, USA: O'Reilly, 2009.
- Huang, Ing-Jer. Despain, Alvin M. "Hardware/Software Resolution of Pipeline Hazards in Pipeline Synthesis of Instruction Set Processors", IEEE, 1993.
- TANENBAUM, Andrew S. Organização estruturada de computadores. São Paulo: Pearson Prentice Hall, 2007, Cap 2.
- MONTEIRO, Mário A. Introdução à organização de computadores. Rio de Janeiro: LTC, 2007, Cap. 1, Cap. 2 e Cap 10.
- MURDOCCA, Miles J. Introdução à arquitetura de computadores. Rio de Janeiro: Campus, 2000 Cap. 8.
- PATTERSON, David A. HENNESSY, John L. Organização e projeto de computadores: a interface hardware/software. 2. ed. Rio de Janeiro: LTC, 2000. 551p.
- STALLINGS, W. Arquitetura e organização de computadores. São Paulo: Prentice Hall, 2002.
- http://www.wiki.org acedido em 27-02-2017

**Sede da Universidade Virtual africana**

The African Virtual University
Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

**Escritório Regional da Universidade Virtual Africana em Dakar**

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org