Organização de Computadores Digitais - 5954008

1. Introdução

Prof. Luiz Otavio Murta Jr

Local: Depto. de Computação e Matemática (FFCLRP/USP)

Principais Tópicos

1. Introdução

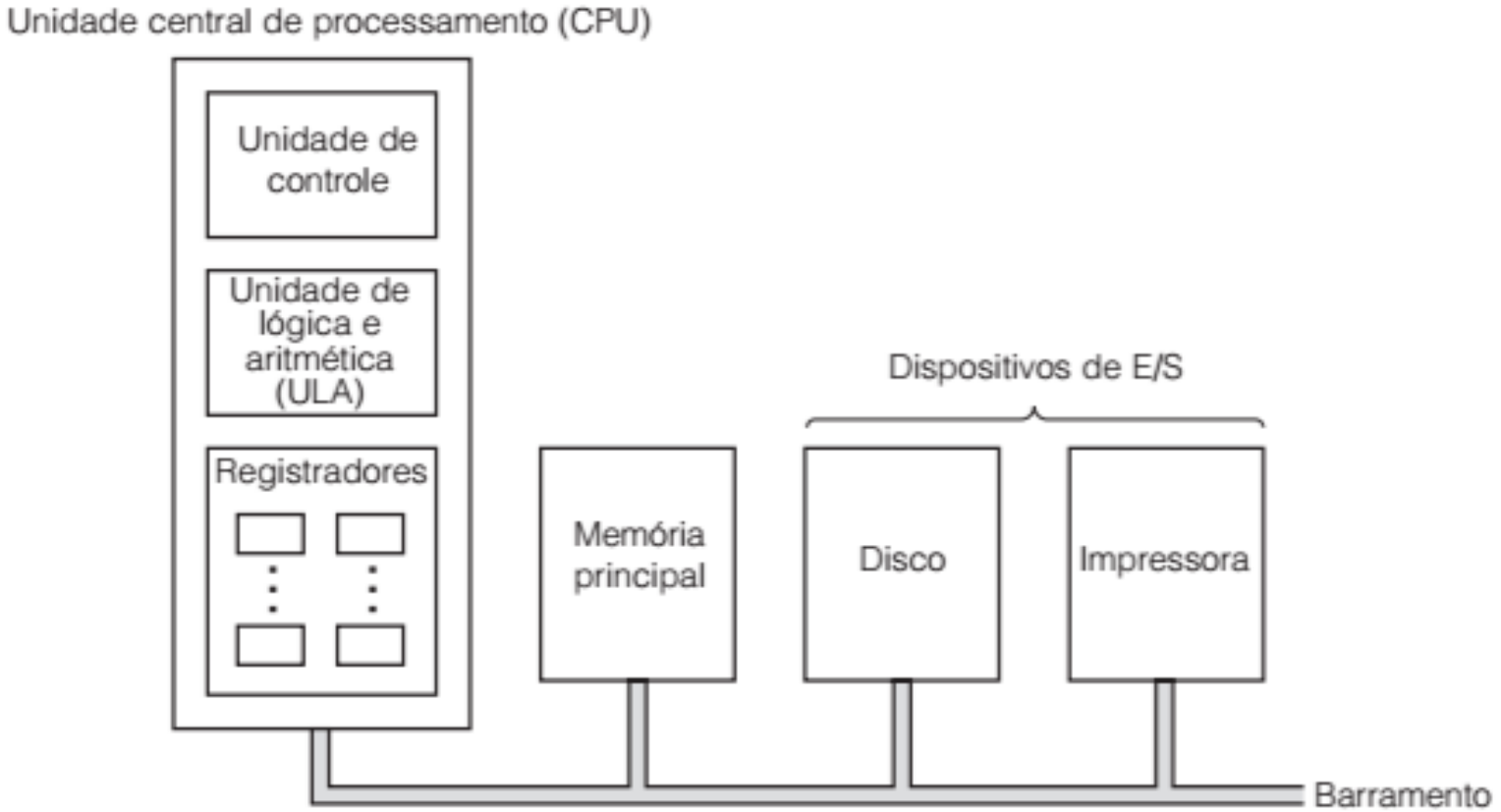
- 1.1. Organização
- 1.2. Breve História da Computação
- 1.3. Informação Digital
  - 1.3.1. Introdução
  - 1.3.2. Codificação da Informação
  - 1.3.3. Sistemas Numéricos
  - 1.3.4. Operações Aritméticas com Números Binários

1.2.5. Processadores

- Um computador digital consiste em um sistema interconectado de processadores, memória e dispositivos de entrada/saída.
- Esta aula é uma introdução a história de parte desses três componentes, sua interconexão e a sua história,
  - como base para o exame mais detalhado de níveis específicos nos cinco capítulos subsequentes.
- Processadores, memórias e dispositivos de entrada/saída são conceitos fundamentais e estarão presentes em todos os níveis,
  - portanto, iniciaremos nosso estudo da arquitetura de computadores examinando todos os três, um por vez.

1.2.5. Processadores

- A organização de um computador simples com uma CPU e dois dispositivos de E/S.



1.2.5. Processadores

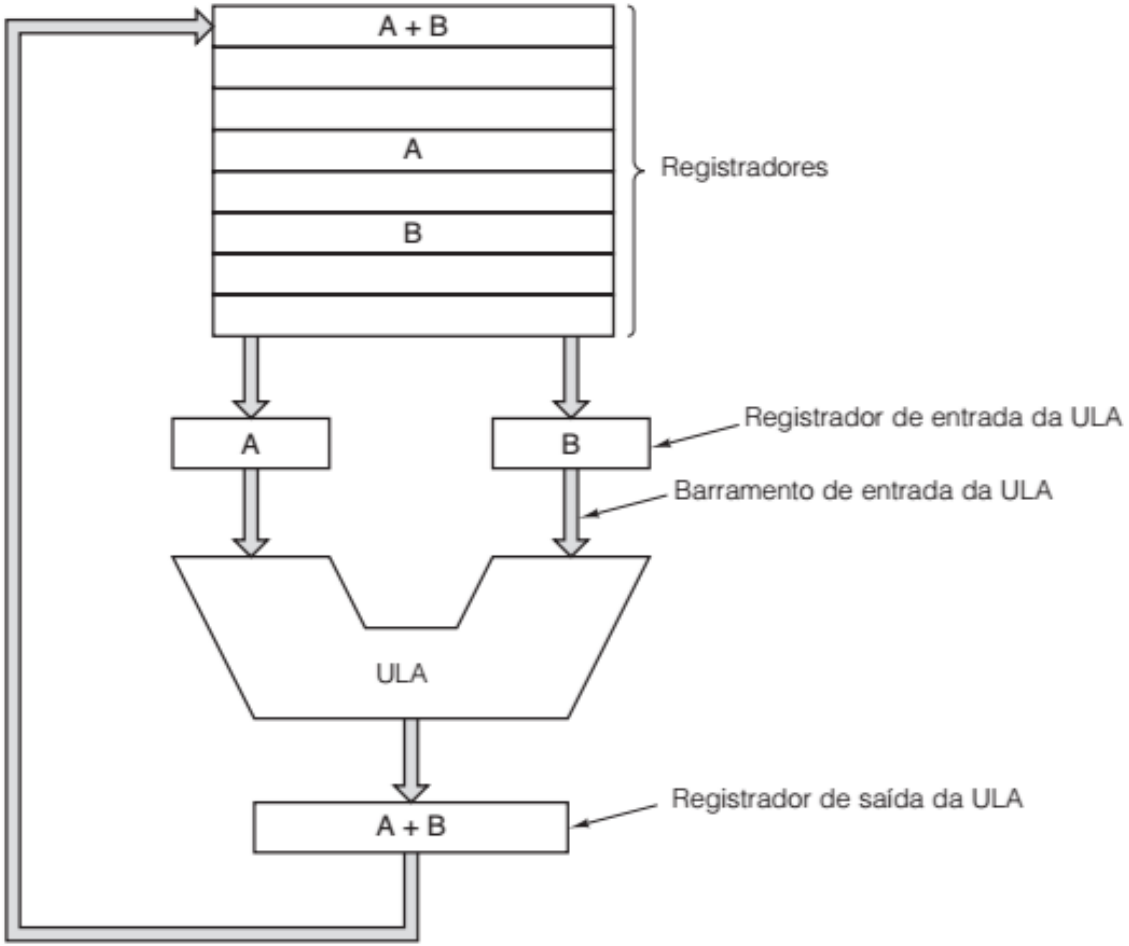
- A CPU é composta por várias partes distintas.
- A unidade de controle é responsável por buscar instruções na memória principal e determinar seu tipo.
- A unidade de aritmética e lógica efetua operações como adição e AND (E) booleano para executar as instruções.
- A CPU também contém uma pequena memória de alta velocidade usada para armazenar resultados temporários e para algum controle de informações.
- Essa memória é composta de uma quantidade de registradores, cada um deles com determinado tamanho e função.
- Em geral, todos os registradores têm o mesmo tamanho.
- Cada um pode conter um número, até algum máximo definido pelo tamanho do registrador.
- Registradores podem ser lidos e escritos em alta velocidade porque são internos à CPU.

1.2.5. Processadores

- O registrador mais importante é o Contador de Programa (PC – Program Counter),
  - que indica a próxima instrução a ser buscada para execução. (O nome “contador de programa” é um tanto enganoso,
  - porque nada tem a ver com contar qualquer coisa; porém, o termo é de uso universal.)
- Também importante é o Registrador de Instrução (IR – Instruction Register),
  - que mantém a instrução que está sendo executada no momento em questão.
- A maioria dos computadores também possui diversos outros registradores, alguns de uso geral, e outros de uso específico.
- Outros registradores são usados pelo sistema operacional para controlar o computador.

1.2.6. Organização da CPU

- O caminho de dados de uma típica máquina de von Neumann.



1.2.6. Organização da CPU

- A ULA efetua adição, subtração e outras operações simples sobre suas entradas,
  - produzindo assim um resultado no registrador de saída,
  - o qual pode ser armazenado em um registrador.
- Mais tarde, ele pode ser escrito (isto é, armazenado) na memória, se desejado.
- Nem todos os projetos têm os registradores A, B e de saída.
- No exemplo, ilustra uma adição, mas as ULAs também realizam outras operações.
- Grande parte das instruções pode ser dividida em uma de duas categorias:
  - registrador-memória, ou registrador-registrador.

1.2.6. Organização da CPU

- Instruções registrador-memória permitem que palavras de memória sejam buscadas em registradores,
  - onde podem ser usadas como entradas de ULA em instruções subsequentes.
- “Palavras” são as unidades de dados movimentadas entre memória e registradores.
- Uma palavra pode ser um número inteiro.
- Outras instruções registrador-memória permitem que registradores voltem à memória para armazenagem.
- O outro tipo de instrução é registrador-registrador.

1.2.6. Organização da CPU

- Uma instrução registrador-registrador típica busca dois operandos nos registradores,
  - traz os dois até os registradores de entrada da ULA,
  - efetua alguma operação com eles
  - (por exemplo, adição ou AND booleano),
  - e armazena o resultado em um dos registradores.
- O processo de passar dois operandos pela ULA e armazenar o resultado é denominado ciclo do caminho de dados e é o coração da maioria das CPUs.
- Até certo ponto considerável, ele define o que a máquina pode fazer.
- Quanto mais rápido for o ciclo do caminho de dados, mais rápido será o funcionamento da máquina.

1.2.7. Execução da instrução

- A CPU executa cada instrução em uma série de pequenas etapas.
- Em termos simples, as etapas são as seguintes:

1. Trazer a próxima instrução da memória até o registrador de instrução.
2. Alterar o contador de programa para que aponte para a próxima instrução.
3. Determinar o tipo de instrução trazida.
4. Se a instrução usar uma palavra na memória, determinar onde essa palavra está.
5. Trazer a palavra para dentro de um registrador da CPU, se necessário.
6. Executar a instrução.
7. Voltar à etapa 1 para iniciar a execução da instrução seguinte.

- Tal sequência de etapas costuma ser denominada ciclo buscar-decodificar-executar. É fundamental para a operação de todos os computadores.

1.2.7. Execução da instrução

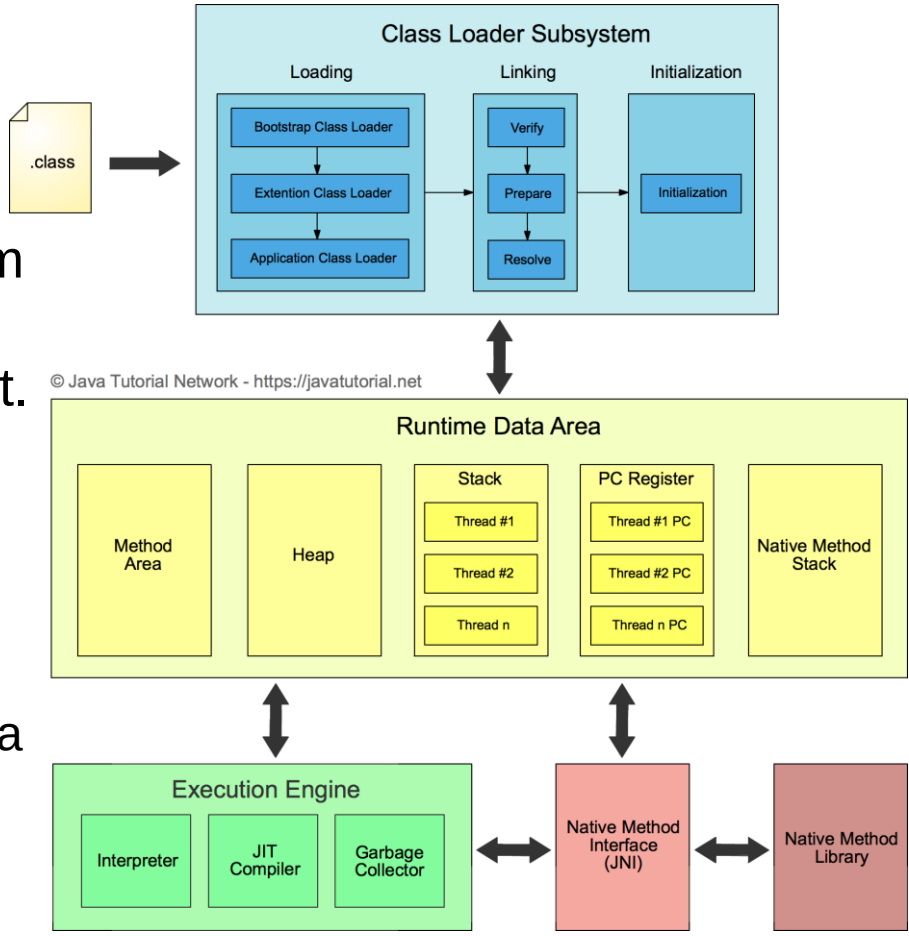
- A CPU executa cada instrução em uma série de pequenas etapas.
- Em termos simples, as etapas são as seguintes:

1. Trazer a próxima instrução da memória até o registrador de instrução.
2. Alterar o contador de programa para que aponte para a próxima instrução.
3. Determinar o tipo de instrução trazida.
4. Se a instrução usar uma palavra na memória, determinar onde essa palavra está.
5. Trazer a palavra para dentro de um registrador da CPU, se necessário.
6. Executar a instrução.
7. Voltar à etapa 1 para iniciar a execução da instrução seguinte.

- Tal sequência de etapas costuma ser denominada ciclo buscar-decodificar-executar. É fundamental para a operação de todos os computadores.

1.2.7. Execução da instrução

- Essa descrição do modo de funcionamento de uma CPU é muito parecida com um programa escrito em inglês.
- A Figura próxima mostra esse programa informal reescrito como um método Java (isto é, um procedimento) denominado interpret.
- A máquina que está sendo interpretada tem dois registradores visíveis para programas usuários:
  - o contador de programa (PC),
  - para controlar o endereço da próxima instrução a ser buscada,
  - e o acumulador (AC), para acumular resultados aritméticos.



1.2.7. Execução da instrução

- Também tem registradores internos para conter:
  - a instrução corrente durante sua execução (instr),
  - o tipo da instrução corrente (instr_type),
  - o endereço do operando da instrução (data_loc),
  - e o operando corrente em si (data).
- Admitimos que as instruções contêm um único endereço de memória.
- A localização de memória endereçada contém o operando, por exemplo, o item de dado a ser somado ao acumulador.

1.2.7. Execução da instrução

- Interpretador para um computador simples (escrito em Java).

```
public class Interp {

    static int PC;                          // contador de programa contém endereço da próxima i
    static int AC;                          // o acumulador, um registrador para efetuar aritmética
    static int instr;                       // um registrador para conter a instrução corrente
    static int instr_type;                  // o tipo da instrução (opcode)
    static int data_loc;                    // o endereço dos dados, ou –1 se nenhum
    static int data;                        // mantém o operando corrente
    static boolean run_bit = true;          // um bit que pode ser desligado para parar a máquina

    public static void interpret(int memory[ ], int starting_ address) {
        // Esse procedimento interpreta programas para uma máquina simples com instruções que têm
        // um operando na memória. A máquina tem um registrador AC (acumulador), usado para
        // aritmética. A instrução ADD soma um inteiro na memória do AC, por exemplo.
        // O interpretador continua funcionando até o bit de funcionamento ser desligado pela instrução HALT.
        // O estado de um processo que roda nessa máquina consiste em memória, o
        // contador de programa, bit de funcionamento e AC. Os parâmetros de entrada consistem
        / na imagem da memória e no endereço inicial.

    PC = starting_address;
    while (run_bit) {
        instr = memory[PC];                 // busca a próxima instrução e armazena em instr
        PC = PC + 1;                        // incrementa contador de programa
        instr_type = get_instr_type(instr); // determina tipo da instrução
        data_loc = find_data(instr, instr_type); // localiza dados (–1 se nenhum)
        if (data_loc >= 0)                  // se data_loc é –1, não há nenhum operando
           data = memory[data_loc];         // busca os dados
        execute(instr_type, data);          // executa instrução
        }
    }
    private static int get_instr_type(int addr) { ... }
    private static int find_data(int instr, int type) { ... }
    private static void execute(int type, int data) { ... }
}
```

1.2.7. Execução da instrução

- Um interpretador subdivide as instruções da máquina em questão em pequenas etapas.
- Por conseguinte, a máquina na qual o interpretador roda deve ser muito mais simples e menos cara do que seria um processador de hardware para a máquina citada.
- Essa economia é bastante significativa se a máquina em questão tiver um grande número de instruções e estas forem razoavelmente complicadas, com muitas opções.
- Basicamente, a economia vem do fato de que o hardware está sendo substituído por software (o interpretador) e custa mais reproduzir hardware do que software.
- Os primeiros computadores tinham conjuntos de instruções pequenos, simples.

1.2.7. Execução da instrução

- Mas a procura por equipamentos mais poderosos levou, entre outras coisas, a instruções individuais mais poderosas.
- Logo se descobriu que instruções mais complexas muitas vezes levavam à execução mais rápida do programa mesmo que as instruções individuais demorassem mais para ser executadas.
- Uma instrução de ponto flutuante é um exemplo de instrução mais complexa.
- O suporte direto para acessar elementos matriciais é outro.
- Às vezes, isso era simples como observar que as mesmas duas instruções muitas vezes ocorriam em sequência, de modo que uma única instrução poderia fazer o trabalho de ambas.

1.2.7. Execução da instrução

- As instruções mais complexas eram melhores porque a execução de operações individuais às vezes podia ser sobreposta ou então executada em paralelo usando hardware diferente.
- No caso de computadores caros, de alto desempenho, o custo desse hardware extra poderia ser justificado de imediato.
- Assim, computadores caros, de alto desempenho, passaram a ter mais instruções do que os de custo mais baixo.
- Contudo, requisitos de compatibilidade de instruções e o custo crescente do desenvolvimento de software criaram a necessidade de executar instruções complexas mesmo em computadores de baixo custo, nos quais o custo era mais importante do que a velocidade.

1.2.7. Execução da instrução

- No final da década de 1950, a IBM (na época a empresa que dominava o setor de computadores) percebeu que prestar suporte a uma única família de máquinas, todas executando as mesmas instruções, tinha muitas vantagens, tanto para a IBM quanto para seus clientes.
- Então, a empresa introduziu o termo arquitetura para descrever esse nível de compatibilidade.
- Uma nova família de computadores teria uma só arquitetura, mas muitas implementações diferentes que poderiam executar o mesmo programa e seriam diferentes apenas em preço e velocidade.
- Mas como construir um computador de baixo custo que poderia executar todas as complicadas instruções de máquinas caras, de alto desempenho?

1.2.7. Execução da instrução

- A resposta foi a interpretação.
- Essa técnica, que já tinha sido sugerida por Maurice Wilkes (1951), permitia o projeto de computadores simples e de menor custo, mas que, mesmo assim, podiam executar um grande número de instruções.
- O resultado foi a arquitetura IBM System/360, uma família de computadores compatíveis que abrangia quase duas ordens de grandeza, tanto em preço quanto em capacidade.
- Uma implementação de hardware direto (isto é, não interpretado) era usada somente nos modelos mais caros.

1.2.7. Execução da instrução

- Computadores simples com instruções interpretadas também tinham outros benefícios, entre os quais os mais importantes eram:

1. A capacidade de corrigir em campo instruções executadas incorretamente ou até compensar deficiências de projeto no hardware básico.
2. A oportunidade de acrescentar novas instruções a um custo mínimo, mesmo após a entrega da máquina.
3. Projeto estruturado que permitia desenvolvimento, teste e documentação eficientes de instruções complexas.

1.2.7. Execução da instrução

- À medida que o mercado explodia em grande estilo na década de 1970 e as capacidades de computação cresciam depressa, a demanda por máquinas de baixo custo favorecia projetos de computadores que usassem interpretadores.
- A capacidade de ajustar hardware e interpretador para um determinado conjunto de instruções surgiu como um projeto muito eficiente em custo para processadores.
- À medida que a tecnologia subjacente dos semicondutores avançava, as vantagens do custo compensavam as oportunidades de desempenho mais alto e as arquiteturas baseadas em interpretador se tornaram o modo convencional de projetar computadores.
- Quase todos os novos computadores projetados na década de 1970, de microcomputadores a mainframes, tinham a interpretação como base.

1.2.7. Execução da instrução

- No final da década de 1970, a utilização de processadores simples que executavam interpretadores tinha se propagado em grande escala, exceto entre os modelos mais caros e de desempenho mais alto, como o Cray-1 e a série Cyber da Control Data.
- A utilização de um interpretador eliminava as limitações de custo inerentes às instruções complexas, de modo que os projetistas começaram a explorar instruções muito mais complexas, em particular os modos de especificar os operandos a utilizar.
- A tendência alcançou seu ponto mais alto com o VAX da Digital Equipment Corporation, que tinha várias centenas de instruções e mais de 200 modos diferentes de especificar os operandos a serem usados em cada instrução.

1.2.7. Execução da instrução

- Infelizmente, desde o início a arquitetura do VAX foi concebida para ser executada com um interpretador, sem dar muita atenção à realização de um modelo de alto desempenho.
- Esse modo de pensar resultou na inclusão de um número muito grande de instruções de valor marginal e que eram difíceis de executar diretamente.
- Essa omissão mostrou ser fatal para o VAX e, por fim, também para a DEC (a Compaq comprou a DEC em 1998 e a Hewlett-Packard comprou a Compaq em 2001).
- Embora os primeiros microprocessadores de 8 bits fossem máquinas muito simples com conjuntos de instruções muito simples, no final da década de 1970 até os microprocessadores tinham passado para projetos baseados em interpretador.

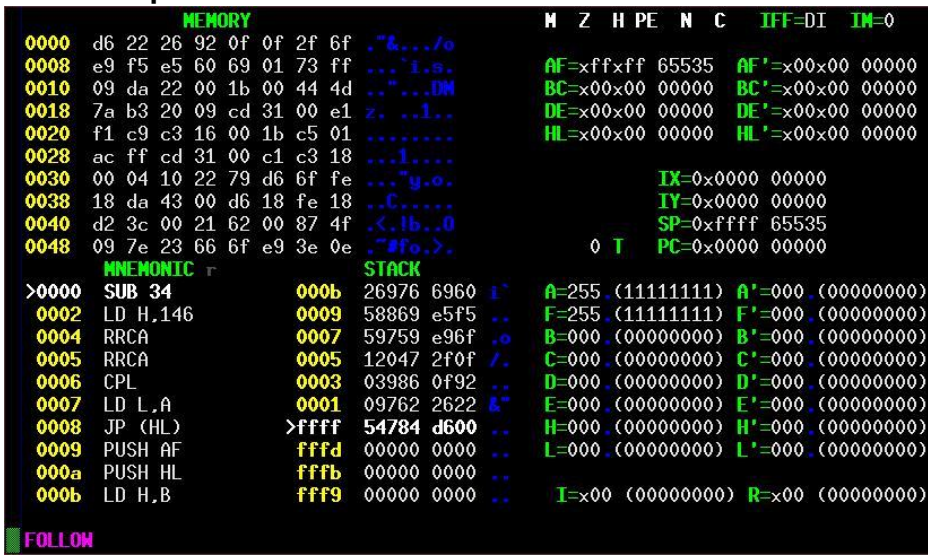
1.2.7. Execução da instrução

- Durante esse período, um dos maiores desafios enfrentados pelos projetistas de microprocessadores era lidar com a crescente complexidade, possibilitada por meio de circuitos integrados.

- Uma importante vantagem do método baseado em interpretador era a capacidade de projetar um processador simples e confinar quase toda a complexidade na memória que continha o interpretador.
- Assim, um projeto complexo de hardware se transformou em um projeto complexo de software.
- O sucesso do Motorola 68000, que tinha um grande conjunto de instruções interpretadas, e o concomitante fracasso do Zilog Z8000 (que tinha um conjunto de instruções tão grande quanto, mas sem um interpretador) demonstraram as vantagens de um interpretador para levar um novo microprocessador rapidamente ao mercado.

1.2.7. Execução da instrução

- Esse sucesso foi ainda mais surpreendente dada a vantagem de que o Zilog desfrutava
  - (o antecessor do Z8000, o Z80, era muito mais popular do que o antecessor do 68000, o 6800).
- Claro que outros fatores também contribuíram para isso, e um dos mais importantes foi a longa história da Motorola como fabricante de chips e a longa história da Exxon
  - (proprietária da Zilog) como empresa de petróleo,
  - e não como fabricante de chips.

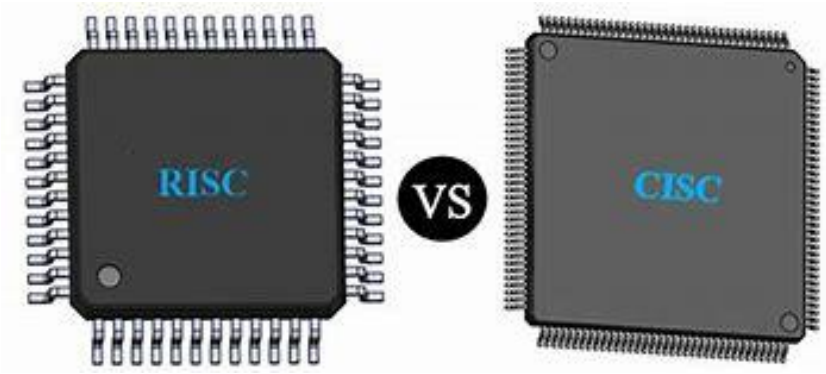


1.2.7. Execução da instrução

- Outro fator a favor da interpretação naquela época foi a existência de memórias rápidas somente de leitura, denominadas memórias de controle, para conter os interpretadores.
- Suponha que uma instrução interpretada típica precisasse de 10 instruções do interpretador, denominadas microinstruções, a 100 ns cada, e duas referências à memória principal a 500 ns cada.
- Então, o tempo total de execução era 2.000 ns, apenas um fator de dois pior do que o melhor que a execução direta podia conseguir.
- Se a memória de controle não estivesse disponível, a instrução levaria 6.000 ns.
- Uma penalidade de fator seis é muito mais difícil de aceitar do que uma penalidade de fator dois.

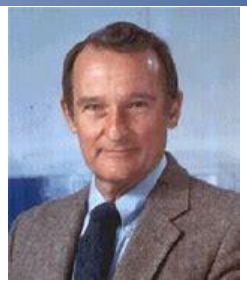
1.2.8. RISC vs. CISC

- Durante o final da década de 1970, houve experiências com instruções muito complexas que eram possibilitadas pelo interpretador.
- Os projetistas tentavam fechar a “lacuna semântica” entre o que as máquinas podiam fazer e o que as linguagens de programação de alto nível demandavam.
- Quase ninguém pensava em projetar máquinas mais simples, exatamente como agora não há muita pesquisa na área de projeto de planilhas, redes, servidores Web etc. menos poderosos (o que talvez seja lamentável).



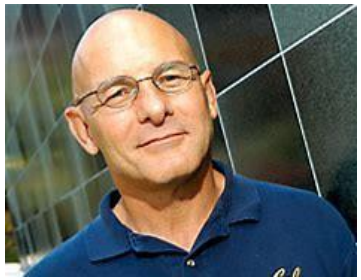
1.2.8. RISC vs. CISC

- Um grupo que se opôs à tendência e tentou incorporar algumas das ideias de Seymour Cray em um minicomputador de alto desempenho foi liderado por John Cocke na IBM.
- Esse trabalho resultou em um minicomputador denominado 801.
- Embora a IBM nunca tenha lançado essa máquina no mercado e os resultados tenham sido publicados só muitos anos depois (Radin, 1982), a notícia vazou e outros começaram a investigar arquiteturas semelhantes.
- Em 1980, um grupo em Berkeley, liderado por David Patterson e Carlo Séquin, começou a projetar chips para CPUs VLSI que não usavam interpretação (Patterson, 1985; Patterson e Séquin, 1982).



Seymour Cray

John Cocke



David Patterson

Carlo Séquin

1.2.8. RISC vs. CISC

- Eles cunharam o termo RISC para esse conceito e deram ao seu chip de CPU o nome RISC I CPU, seguido logo depois pelo RISC II.
- Um pouco mais tarde, em 1981, do outro lado da baía de São Francisco, em Stanford, John Hennessy projetou e fabricou um chip um pouco diferente, que ele chamou de MIPS (Hennessy, 1984).
- Esses chips evoluíram para produtos de importância comercial, o SPARC e o MIPS, respectivamente.
- Esses novos processadores tinham diferenças significativas em relação aos que havia no comércio naquela época.
- Uma vez que essas novas CPUs não eram compatíveis com os produtos existentes, seus projetistas tinham liberdade para escolher novos conjuntos de instruções que maximizassem o desempenho total do sistema.

John Hennessy

1.2.8. RISC vs. CISC

- Embora a ênfase inicial estivesse dirigida a instruções simples, que podiam ser executadas rapidamente, logo se percebeu que projetar instruções que podiam ser emitidas (iniciadas) rapidamente era a chave do bom desempenho.
- Na verdade, o tempo que uma instrução demorava importava menos do que quantas delas podiam ser iniciadas por segundo.
- Na época em que o projeto desses processadores simples estava no início, a característica que chamou a atenção de todos era o número relativamente pequeno de instruções disponíveis, em geral cerca de 50.
- Esse número era muito menor do que as 200 a 300 de computadores como o VAX da DEC e os grandes mainframes da IBM.

VAX
(CISC)

1.2.8. RISC vs. CISC

- De fato, o acrônimo RISC quer dizer Reduced Instruction Set Computer (computador com conjunto de instruções reduzido).
- Em comparação com CISC, que significa Complex Instruction Set Computer (computador com conjunto de instruções complexo).
- Hoje em dia, poucas pessoas acham que o tamanho do conjunto de instruções seja um assunto importante, mas o nome pegou.
- Encurtando a história, seguiu-se uma grande guerra santa, com os defensores do RISC atacando a ordem estabelecida (VAX, Intel, grandes mainframes da IBM).
- Eles afirmavam que o melhor modo de projetar um computador era ter um pequeno número de instruções simples que executassem em um só ciclo do caminho de dados.

1.2.8. RISC vs. CISC

- Ou seja, buscar dois registradores, combiná-los de algum modo (por exemplo, adicionando-os ou fazendo AND) e armazenar o resultado de volta em um registrador.
- O argumento desses pesquisadores era de que, mesmo que uma máquina RISC precisasse de quatro ou cinco instruções para fazer o que uma CISC fazia com uma só, se as instruções RISC fossem dez vezes mais rápidas (porque não eram interpretadas), o RISC vencia.
- Também vale a pena destacar que, naquele tempo, a velocidade de memórias principais tinha alcançado a velocidade de memórias de controle somente de leitura, de modo que a penalidade imposta pela interpretação tinha aumentado demais, o que favorecia muito as máquinas RISC.

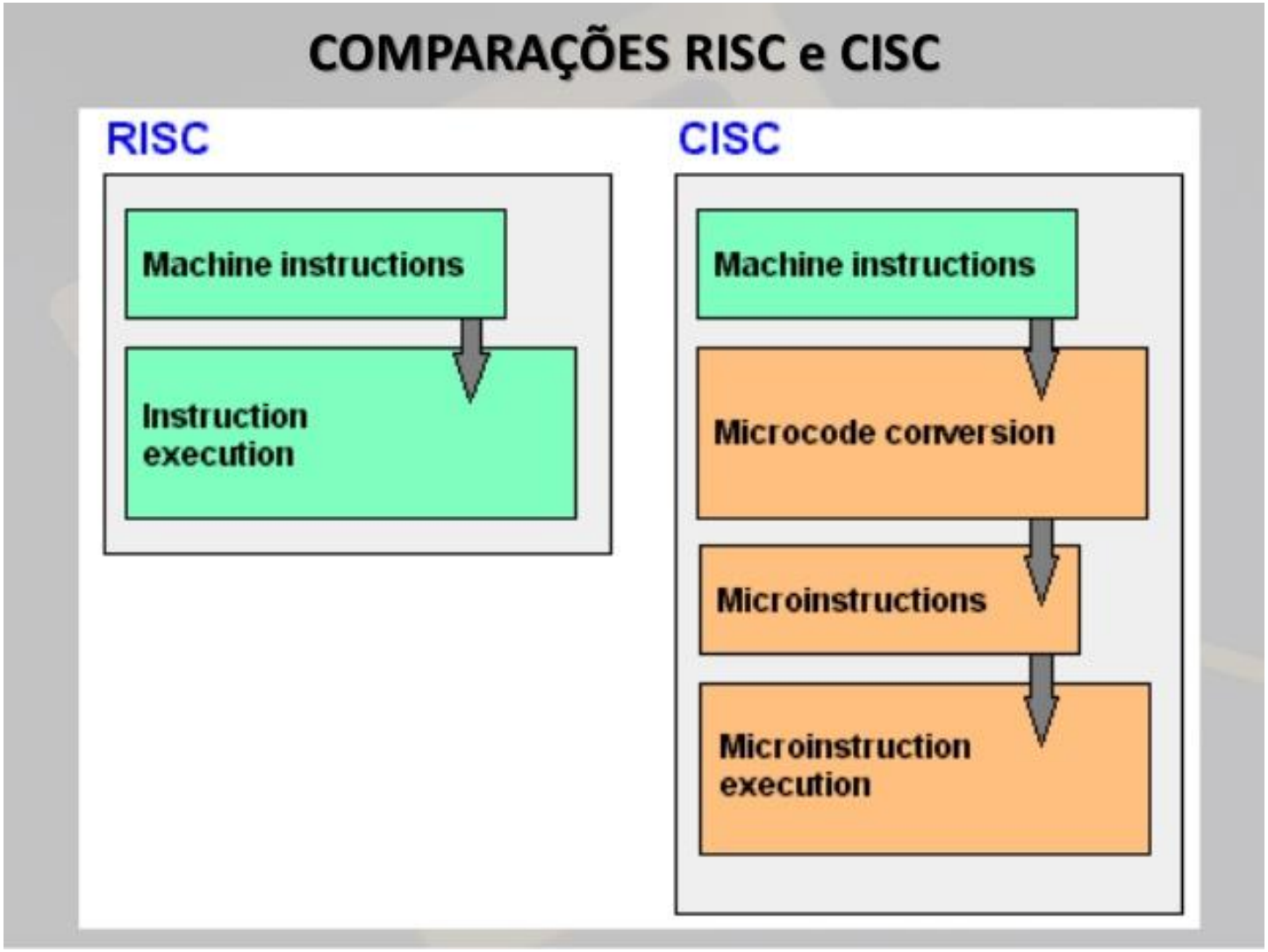
1.2.8. RISC vs. CISC

- Era de imaginar que, dadas as vantagens de desempenho da tecnologia RISC, as máquinas RISC (como a Sun UltraSPARC) passariam como rolo compressor sobre as máquinas CISC (tal como a Pentium da Intel) existentes no mercado.
- Nada disso aconteceu.
- Por quê?
- Antes de tudo, há a questão da compatibilidade e dos bilhões de dólares que as empresas tinham investido em software para a linha Intel.
- Em segundo lugar, o que era surpreendente, a Intel conseguiu empregar as mesmas ideias mesmo em uma arquitetura CISC.
- A partir do 486, as CPUs da Intel contêm um núcleo RISC que executa as instruções mais simples (que normalmente são as mais comuns) em um único ciclo do caminho de dados, enquanto interpreta as mais complicadas no modo CISC de sempre.

1.2.8. RISC vs. CISC

- O resultado disso é que as instruções comuns são rápidas e as menos comuns são lentas.
- Mesmo que essa abordagem híbrida não seja tão rápida quanto um projeto RISC puro, ela resulta em desempenho global competitivo e ainda permite que softwares antigos sejam executados sem modificação.

1.2.8. RISC vs. CISC



1.2.8. A favor do CISC

- **Lógica cabeada levando ao controle do microcódigo**
  - Supostamente mais fácil extensibilidade
- **Ajuste de Performance**
  - Implementações em hardware de algumars funções de alto nível
- **Marketing**
  - Permite adicionar instruções consideradas bem sucedidas de competidors
  - “hype” em novas características
  - Compatibilidade: somente novas extensãoe são possíveis

1.2.8. Problemas do CISC

- **Ajuste performance mal successido**
  - Raramente usou instruções de alto nível
  - Algumas vezes mais mais lento que a sequência equivalente
- **Alta complexidade**
  - Gargalos nos pipelines levando a taxas de clock mais baixas
  - Gerenciamento de iterrupções pode complicar ainda mais a situação
- **Marketing**
  - Os tempos de projetos e erros frequentes no microcódigos prejudicam a competitividade

1.2.8. Características RISC

- **Baixa complexidade**
  - Generalmente resulta em aceleração em termos gerais
  - Implementação da lógica programada em hardware, ou microcódigos menos sujeita a erros
- **Vantagens na implementação VLSI**
  - Menos transistors necessários
  - Deixando espaços extra para mais registradores, e memórias caches
- **Marketing**
  - Tempos de projetos reduzida, menor possibilidade de erros, e mais opções aumentam a competitividade no mercado

1.2.8 Problemas com compiladores RISC

- **Os compiladores são**
  - Computacionamente mais complexos
  - Porém mais portáveis

- **Os compiladores escrevem**
  - Menos instruções $\rightarrow$ com tarefas provavelmente mais fáceis
  - Instruções mais simples $\rightarrow$ com provavelmente menos bugs
  - Podem reusar técnicas de otimização

1.2.8 *concepções* RISC vs. CISC

- **Argumentos favorecendo o RISC: projeto simples, tempo curto de projeto, velocidade, preço…**

- **Estudo do RISC deve incluir compromisso hardware/software, fatores influenciando a performance dos computadores e avaliação do lado da industria.**

1.2.8 *concepções* RISC vs. CISC

- **Implicações incorretas dos dois acrônimos: RISC e CISC.**
  - Eles não são bifurcações entre as quais projetistas têm que escolher

- **Descuidadosamente deixando de for a "participação" dos Sistemas Operacionais**

1.2.8 *concepções* RISC vs. CISC

- **Tempo de projeto reduzido?**
  - acadêmico <-> industrial

- **Performance claims of RISC proponent do not decouple design features like MRSs.**
  - MRSs can have a remarkable effect on program execution

1.2.8 Concluão – RISC vs. CISC?

- **CISC**
  - *Efetivamente realiza um* Sistema de Computação de uma Linguagems de Alto Nível ***particular*** *em Hardware* – com recurrendo a **custos de desenvolvimento de Hardware** quando mudanças são necessárias

- **RISC**
  - *Permite a efetiva realização de* ***qualquer*** Sistema de Computação de Alta Nível *em Software* – recurrendo a **custos de desenvolvimento de Software** quando mudanças são necessárias

1.2.8 Concluão – RISC vs. CISC?

- **Solução híbrida**
  - RISC core & CISC interface
  - Ainda existe *ajuste performance específica*

- **ISA (instruction set architecture) ótima**
  - Entre RISC & CISC
  - Pouco, cuidadosamente escolhida, instruções complexas útéis
  - Ainda tem *problemas de gerenciamento de complexidade*