Faculdade de Computação

Arquitetura e Organização de Computadores 1

Laboratório de Programação Assembly 1

MEMORIAL DESCRITIVO DE SOLUÇÕES DA DISCIPLINA DE ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES 1
DESENVOLVIMENTO E ANÁLISE DE ALGORITMOS NA ARQUITETURA IAS MACHINE E HEXADECIMAL

Ana Alice Cordeiro de Souza, Anna Karolyna Pereira Santos, Fernanda Ferreira de Melo, Sthephanny Caroline da Silva Santos

Uberlândia/MG, 2024

1. Introdução

Este documento apresenta o desenvolvimento de algoritmos para a **Arquitetura IAS Machine**, uma das primeiras implementações do conceito de "programa armazenado", idealizada por **John Von Neumann** no Instituto de Estudos Avançados em Princeton. A máquina IAS é a base de praticamente todos os computadores modernos, sendo essencial para o desenvolvimento da computação como conhecemos hoje.

A IAS Machine possuía uma arquitetura simples, mas inovadora, com uma **memória de 1024 palavras de 40 bits**, sendo que cada palavra poderia armazenar **dois comandos de 20 bits**. A instrução era composta por um **opcode de 8 bits** e um **endereço de 12 bits**, o que limitava as operações possíveis, mas fornecia uma base sólida para o armazenamento de programas e dados na mesma memória.

O trabalho desenvolvido consiste na construção de algoritmos para resolver uma lista de problemas utilizando a **linguagem de montagem** da IAS e sua correspondente **representação em hexadecimal**. A execução dos códigos foi simulada por meio de um **aplicativo de simulação**, fornecido pelo canal da equipe AOC1 na plataforma **MS Teams**.

2. Objetivo

O objetivo principal deste trabalho é desenvolver soluções que explorem a **Arquitetura IAS**, criando sequências de instruções em **assembly** e convertendo-as para **hexadecimal**. Essas soluções devem ser capazes de resolver problemas específicos, utilizando a lógica e o conjunto de instruções limitadas da IAS Machine. Além disso, busca-se reforçar o conceito de **programa armazenado**, simulando a execução dos algoritmos em um ambiente digital.

3. Desenvolvimento dos Algoritmos

Os algoritmos foram desenvolvidos utilizando a linguagem de montagem da IAS, em que cada operação foi cuidadosamente planejada para resolver os problemas propostos.

3.1. Problema 1

Escreva duas sequências de códigos da Arquitetura IAS para resolver o problema descrito, escreva um algoritmo com uma lógica de programação ruim (péssimo desempenho) e, outro algoritmo com uma lógica otimizada. Para avaliar o desempenho utilize a métrica da quantidade de instruções executadas. Para simplificar e uniformizar a codificação, considere que as variáveis escalares necessárias para a solução do problema estarão armazenadas a partir do endereço de memória 02016 e os vetores idades e somas nos endereços 03016 e 05016, respectivamente. Construa o algoritmo em linguagem de montagem do IAS, traduza o código para a representação hexadecimal correspondente e simule a execução no aplicativo de simulação IAS Machine.

Solução:

Passo 1: Inicialização O programa começa carregando a primeira idade e o número de alunos em registradores temporários. Esses valores serão utilizados para iniciar o processo de soma e armazenar os resultados.

Passo 2: Loop Principal O código entra em um loop onde, para cada aluno, ele:

1. Carrega a idade do aluno atual.
2. Soma todas as outras idades, exceto a do aluno atual.
3. Armazena a soma no vetor de resultados.

Passo 3: Armazenamento dos Resultados Para cada iteração do loop, o programa armazena a soma calculada para o aluno corrente no vetor de resultados, mantendo o processo até que todas as idades tenham sido processadas.

Comandos com comentários:

LOAD M(010), STOR M(015)

Carrega a primeira idade do vetor e a armazena temporariamente.

LOAD M(007), STOR M(020)

Carrega o número total de alunos e armazena esse valor para controle do loop.

LOAD M(015), ADD M(011)

Carrega a primeira idade e soma com a próxima idade no vetor.

SUB M(015), STOR M(012)

Subtrai a idade atual (do aluno) do somatório e armazena o resultado no vetor de resultados.

LOAD M(012), ADD M(011)

Carrega a próxima idade e a soma ao acumulador para o próximo cálculo.

STOR M(018), LOAD M(000)

Armazena o resultado da soma no vetor de resultados e zera o acumulador para a próxima operação.

LOAD M(010), SUB M(015)

Carrega a próxima idade e subtrai a idade do aluno atual.

STOR M(020), LOAD M(000)

Armazena o resultado no vetor de resultados e zera o acumulador novamente.

LOAD M(018)

Carrega a próxima idade do vetor e incrementa o índice para o próximo aluno.

STOR M(010)

Armazena o valor no vetor e finaliza o loop, encerrando o programa.

3.2. Problema 2

Para simplificar e uniformizar a codificação, considere que as variáveis escalares necessárias para a solução do problema estarão armazenadas a partir do endereço de memória 02016 e os valores da sequência a partir do endereço 03016. Escreva sequências de códigos da Arquitetura IAS para resolver o problema descrito. Construa o algoritmo em linguagem de montagem do IAS, traduza o código para a representação hexadecimal correspondente e simule a execução no aplicativo de simulação IAS Machine.

Solução:

O programa foi desenvolvido para resolver a conjectura de Collatz em linguagem de montagem para a arquitetura IAS. Ele começa carregando o valor nnn da memória e o armazena temporariamente. A partir desse valor inicial, o código verifica se *n* é par ou ímpar. Caso *n* seja par, o programa divide o número por 2; se for ímpar, multiplica-o por 3 e adiciona 1. Após realizar a operação correspondente, o novo valor de *n* é atualizado e armazenado na memória, e o resultado é gravado em uma posição de memória consecutiva, que representa a sequência de Collatz.

O processo é controlado por um loop, que continua até que o valor de *n* seja igual a 1. Durante cada iteração, o programa repete as verificações e cálculos, garantindo que os valores da sequência sejam corretamente armazenados. Assim que o número *n* se torna 1, o programa finaliza a execução, armazenando o valor final na memória.

A lógica de controle é simples: são utilizadas instruções de carregamento e armazenamento para manipular o valor de *n*, além de operações aritméticas de divisão e multiplicação para aplicar as regras da conjectura. O fluxo do programa é controlado por instruções de salto que permitem a repetição do cálculo até que o critério de parada (quando $n = 1$) seja atingido.

Por exemplo, se o valor de *n* fornecido for 6, o programa segue os seguintes passos: carrega o valor 6, verifica que é par e o divide por 2, obtendo 3. Em seguida, como 3 é ímpar, multiplica-o por 3 e adiciona 1, resultando em 10. O processo continua com 10, 5, 16, 8, 4, 2, até que o valor chegue a 1. Toda a sequência, que no caso de $n = 6$ seria 6, 3, 10, 5, 16, 8, 4, 2, 1, é gravada consecutivamente na memória.

3.3. Problema 3

A potenciação ou exponenciação (x^n) é uma das operações básicas no universo dos números naturais onde um dado número x é multiplicado por ele mesmo, uma quantidade n de vezes. O fragmento 1 apresenta um algoritmo simples para calcular x n (x elevado a n).

Fragmento 1	

`int expo1(int x, int n){`
`  int result = 1;`
`  while (n>0){`
`  result *= x;`
`  n--;`
`  }`
`  return result;`
`}`
	Embora esse algoritmo seja relativamente eficiente, com desempenho em tempo linear ou *O(n)*, ele pode ser aprimorado. Poderíamos fazer a mesma tarefa em *O(log (n) + log (n))*. De que maneira? Usando um método chamado ***exponenciação quadrática***.

Solução:

Para começar, foi feito o que seria equivalente a declaração de variáveis em código C. Esses espaços na memória ("variáveis") vão ser utilizados para guardar os números com os quais serão realizadas as operações. Então, ele escreve na memória todos os valores para o meu armazenamento.

- Uma linha para a base (linha 020)

- Uma linha para expoente (linha 021)

- Uma linha com o resultado (linha 022)

- Uma linha para o número 1, isso porque vai ser o valor decrementado na hora de usar o loop (linha 023)

(Linha 000)

Faz um LOAD MQ(X) no endereço do meu resultado (que por hora é 1) e vai multiplicar lá no endereço da base (x = 2). O valor da multiplicação está em MQ (equivalente a linha 4 do código C. Como o valor tá em MQ, usar o comando para levar para AC para pode escreve-lo em outro lugar da memória.

(Linha 001)

Aqui é realizada a operação que transfere o valor de MQ para AC. Depois a máquina é instruída para utilizar o comando STOR M(X) para o endereço de resultado.

(Linha 002)

Carregou o valor do expoente("n", guardado em 021) para poder decrementar o valor 1, guardado na linha 023. (equivalente a linha 5 do código C). O resultado da subtração entre o expoente e 1 está no registrado AC (Usar o STOR). Quero escrever ele em expoente porque ele vira o novo expoente.

Obs: Antes de utilizar a instrução JUMP + M(X,0:19), eu preciso decrementar 1 para que ele, lá na frente, não faça x^5 (considerando que foi definido expoente = 4). Então dou a instrução de decrementar antes. Mas aqui o programa não faz a função de escrever o novo expoente, ele vai servir apenas para a função JUMP conseguir analisar de forma correta se o expoente é negativo ou não.

(Linha 003)

Agora, para fazer com que o laço funcione, vamos usar os JUMPs. O expoente é quem me indicar se meu programa vai continuar rodando ou não, pois ele tem que ser positivo. O comando a seguir é para voltar para o início do programa(no caso do programa em C, seria voltar para o início do comando while).

0F -> comando de JUMP condicional em hexadecimal

Eu uso o salto condicional que verifica se o número é positivo ou não (nesse caso to verificando o expoente). Uma vez que ele faz isso, caso positivo, ele usa o jump condicional. Ele volta para a linha que eu indiquei (nesse caso a 000) e analisa as duas instruções: 09 ou 0B. Como preciso mandar o número para o registrador MQ, para fazer a multiplicação tudo de novo, então uso o salto condicional para a esquerda.

Fragmento 2

Ideia básica: para qualquer x^n , se a n for par, poderíamos escrevê-lo como (x ^ n / 2) ^ 2. Se n for ímpar, por outro lado, poderíamos escrevê-lo como x * (x ^ (n-1/2)) ^ 2. Veja a figura abaixo

$$x^n = \begin{cases} 1, & \text{if } n = 0 \\ \frac{1}{x}^{-n}, & \text{if } n < 0 \\ x \cdot \left(x^{\frac{n-1}{2}}\right)^2, & \text{if } n \text{ is odd} \\ \left(x^{\frac{n}{2}}\right)^2, & \text{if } n \text{ is even} \end{cases}$$	Uma versão iterativa do ***algoritmo de exponenciação quadrática*** é mostrado no Fragmento 2 onde, em cada etapa, divide-se o expoente por dois e eleva ao quadrado a base e, nas iterações em que o expoente é ímpar, você multiplica o resultado pela base.

Fragmento 2	

`int expo2(int x, int n){`
`  int result = 1;`
`  while (n){`
`  if (n%2==1){`
`  result *= x;`
`  }`
`  n /= 2;`
`  x *= x;`
`  }`
`  return result;`
`}`
	Tarefa: Escreva em linguagem de montagem do IAS os dois algoritmos (Fragmento 1 e Fragmento 2). Faça uma análise de desempenho dos dois algoritmos e descreva os resultados obtidos. Para facilitar a análise, contabilize o número de instruções executadas para um valor de n grande.

Solução:

Para começar, nesse outro fragmento também foi feito o que seria equivalente a declaração de variáveis em código C que serão utilizados da mesma maneira, para guardar os números com os quais serão realizadas as operações. Então, ele escreve na memória todos os valores para o meu armazenamento.

- Uma linha para a base (linha 020)

- Uma linha para expoente (linha 021)

- Uma linha com o resultado (linha 022)

- Uma linha para o número 2, que será utilizado como divisor do expoente (linha 023)

Em seguida, a primeira instrução:é carregado o valor do endereço de memória 021 (expoente n) para o acumulador (AC) e é realizada a divisão pelo valor no endereço 023 (que contém o valor 2), em seguida, é dividido por 2.

Em seguida, é carregado o valor do endereço de memória que contém o expoente (n) para o acumulador (AC) e realiza a divisão pelo valor no endereço que contém o valor 2. Esta divisão é essencial para a técnica de exponenciação por quadrados. Em seguida, o código subtrai o valor 1 do acumulador. Se o resultado for ímpar (ou seja, a subtração não resulta em zero), o controle é transferido para uma parte do código que lida com a multiplicação de x pelo valor intermediário.

Depois de verificar se a paridade do expoente (while (n)), o código continua a dividir o expoente por 2, recarregando o valor e atualizando conforme precisar. O valor da base (x) é carregado e multiplicado por si mesmo, seguindo a técnica de exponenciação por quadrados. O resultado dessa multiplicação é então armazenado de volta no endereço de memória correspondente a x.

O loop principal continua a verificar o valor do expoente. Se o expoente atingir zero, o loop será encerrado. Até que isso aconteça, ele retorna ao início do processo, fazendo com que as operações de divisão e multiplicação continuem.

Se o expoente original for ímpar, o código executa uma multiplicação adicional. Ele carrega o valor intermediário (resultado parcial) e multiplica pelo valor de x. O resultado dessa operação é então armazenado como o novo valor do resultado final.

O código implementa a exponenciação por quadrados para calcular x^n. Ele verifica se o expoente é ímpar ou par, divide o expoente por 2 iterativamente, e multiplica x por si mesmo conforme necessário. Se o expoente é ímpar, uma multiplicação adicional é realizada. O controle de fluxo é gerido através de saltos condicionais e incondicionais, permitindo que o algoritmo continue até que o expoente seja reduzido a 0, momento em que o cálculo é concluído.

3.4. Problema 4

Escreva em linguagem de máquina do IAS um programa que faça a classificação em pares ou ímpares de valores armazenados em um vetor V localizados em memória a partir da posição 100. O programa deve armazenar os valores classificados como pares no vetor "pares" localizado na posição de memória 110 e os valores classificados como ímpares no vetor "ímpares" localizado na posição de memória 120.

Fragmento 3

```
int main( ){
     int V[10], pares[10], impares[10];
     int i, j=0, k=0;
     for(i=0;i<10;++i) {
          if (V[i]%2==0) {
               pares[j] = V[i];
               j = j + 1;
          }else {
               impares[k] = V[i];
               k = k + 1;
          }
     }
     return 0;
}
```

Tarefa: Escreva sequências de códigos da Arquitetura IAS para resolver o problema de classificação (Fragmento 3). Construa o algoritmo em linguagem de montagem do IAS, traduza o código para a representação hexadecimal correspondente e simule a execução no aplicativo de simulação IAS Machine.

Solução:

O código Assembly tem o objetivo de processar um vetor de 10 elementos, separando números pares e ímpares em diferentes áreas da memória. Primeiro, há atribuição dos valores 1, 4, 7, 14, 19, 22, 25, 30, 35 e 47 no vetor V presente na posição 100. Em seguida, criamos um índice de valor 9 na posição 130, ele nos ajudará no controle de números restantes.

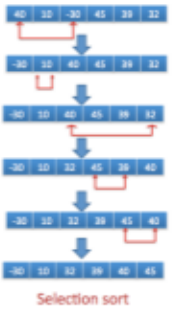
Em suma, o código percorre o vetor V, determinando se cada número é par ou ímpar. Com isso, o código organiza os números conforme solicitado. Quando todos os elementos são processados, o programa finaliza, deixando os números devidamente separados e alocados na memória.

3.5. Problema 5

Escreva em linguagem de montagem do IAS um programa que realize a ordenação de elementos de um vetor V de inteiros, em ordem crescente. A ordenação deve ser feita no próprio vetor V, sem utilizar um vetor auxiliar. Considere que os elementos do vetor V estão armazenados a partir da posição 10016 da memória.

```
int main(){
 int n = 10, i, j, aux;
 int v[10] = {3,7,5,8,0,1,6,9,4,2};

 for(i=0; i<n-1; i++){
     for(j=i+1; j<n; j++){
         if(v[j]<v[i]) {
             aux = v[i];
             v[i] = v[j];
             v[j] = aux;
         }
     }
 }
}
```



Solução:

O programa foi desenvolvido para ordenar um vetor de inteiros em linguagem de montagem para a arquitetura IAS. Ele realiza a ordenação de forma direta, sem utilizar um vetor auxiliar. O código começa carregando o valor inicial ‘i’ e ‘j’, que são utilizados para controlar os laços de repetição.

Primeiramente, o valor de ‘i’ é carregado da memória e armazenado para inicializar o laço externo. Em seguida, o valor de ‘j’ é carregado para iniciar o laço interno. O programa então compara os elementos do vetor ‘v’ que estão armazenados nas posições de memória a partir de 100. Se o valor de ‘v[j]’ for menor que ‘v[i]’, o código realiza a troca dos valores. O valor de ‘v[i]’ é armazenado temporariamente, depois ‘v[j’] é movido para ‘v[i]’, e o valor temporário é movido para ‘v[j]’.

Os laços continuam incrementando os índices ‘i’ e ‘j’ até que todos os elementos do vetor sejam ordenados em ordem crescente. O controle do fluxo é feito por meio de instruções de salto, que permitem a repetição das comparações e trocas até que o vetor esteja completamente ordenado. Quando ‘i’ e ‘j’ são incrementados e os laços terminam, o programa finaliza a execução.

Por exemplo, para o vetor inicial {3, 7, 5, 8, 0, 1, 6, 9, 4, 2}, o programa processa e ordena os elementos da seguinte forma: primeiro, ‘3’ é comparado com os outros elementos e trocado conforme necessário, depois o próximo valor é processado, e assim por diante. O resultado final, após a execução do programa, é o vetor ordenado {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.

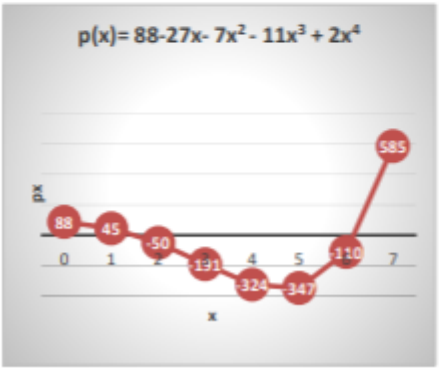
O fluxo do programa é controlado pelas instruções de carregamento, armazenamento, comparação e salto, garantindo que a ordenação seja feita no próprio vetor sem a necessidade de espaço adicional para um vetor auxiliar.

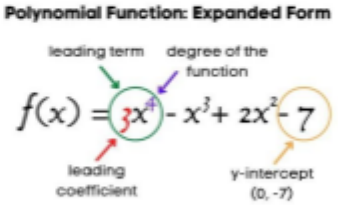
3.6. Problema 6

Escreva em linguagem de montagem do IAS um programa que calcule o valor do polinômio p(x)=a0+a1x+...+anx n em k pontos distintos (valores de x). O programa receberá os valores de n (ordem do polinômio), dos coeficientes reais do polinômio (a0, a1, ..., an), a quantidade

de pontos k para o cálculo de p(x) e os k pontos de teste (x1, x2, ..., xk). Dica: um polinômio de ordem n pode ser representado pelos coeficientes guardados em um vetor de tamanho n+1 elementos.

```
// p(x)= 88-27x- 7x² - 11x³ + 2x⁴
int main() {
  int i, j, k = 8, n = 4;
  int coef[5]={88,-27,-7,-11,2};
  int x[8] = {0,1,2,3,4,5,6,7};
  int Px[8]={0};
  for(i=1;i<=k;i++){
      for(j=0;j<=n;j++) Px[i] += coef[j]*pow(x[i],j);
  }
}
```





Solução:

O algoritmo desenvolvido tem como objetivo calcular o valor de um polinômio em múltiplos pontos de teste, utilizando uma abordagem simples e eficiente. O programa inicia solicitando ao usuário que insira o grau do polinômio *n*, os coeficientes a0,a1,…,a*n*, e a quantidade de pontos de teste k, juntamente com os valores de x*1*,x*2,*…,x*k* nos quais o polinômio será avaliado.

Após a coleta dos dados, o programa utiliza a fórmula do polinômio:

$$p(x)=a_0+a_1x+...+a_nx^n$$

Para cada valor de x*i*, o programa realiza o cálculo do polinômio através de um laço iterativo que soma os termos individualmente, levando em consideração os coeficientes e as potências correspondentes de cada valor de x*i*. A implementação do cálculo é direta, seguindo o formato tradicional de avaliação de polinômios.

Após os cálculos, os resultados são armazenados em uma estrutura de dados e exibidos ao usuário. O programa também permite o uso desses valores para outras análises ou operações subsequentes, conforme necessário.

4. Conclusão

O desenvolvimento deste trabalho permitiu uma maior compreensão da arquitetura clássica proposta por **John Von Neumann** e seu impacto nos sistemas computacionais modernos. A programação na **IAS Machine** destacou a importância do conceito de **programa armazenado** e a eficiência do conjunto limitado de instruções.

Além disso, o exercício de converter os algoritmos para **hexadecimal** proporcionou uma visão mais aprofundada sobre a forma como as instruções são manipuladas pela máquina em nível de hardware. A simulação dos códigos em um ambiente controlado possibilitou a validação das soluções e garantiu a exatidão dos resultados.