Universidade Federal do Rio de Janeiro

Aluno: Gabriel Aureo de Oliveira Campos **DRE:** 115111736
Prof. Gabriel Pereira Silva

Arquitetura de Computadores - 2018/2

1) a. O modelo de von Neumann foi inovador por ter proposto um modelo no qual os programas pudessem ser armazenados na mesmo espaço de memória dos dados, podendo assim, manipulá-los.

b. A Arquitetura de Harvard baseia-se na separação do armazenamento e barramento de dados e instruções. Graças aos barramentos exclusivos, um computador utilizando a arquitetura de Harvard pode acessar um dado e uma instrução simultaneamente, o que não é possível na arquitetura von Neumann.

c. O barramento único, que subdivide o computador em 3 componentes principais e os conecta: **processador** (que integra ULA, registradores e unidade de controle), **memória** e **unidade de entrada/saída**. O barramento único exerce as seguintes funções:

- **Barramento de dados:** Transporta instruções dos programas e dados.
- **Barramento de endereços:** Transporta endereços de posições a serem acessadas pela memória.
- **Barramento de controle:** Realiza a ponte entre a CPU e outros dispositivos no computador, carregando comandos da CPU e recebendo sinais de status dos dispositivos.

d. Não. Muitos microcomputadores antigos usavam apenas uma instância do barramento de sistema. Em 1981, porém, a IBM adotou a arquitetura ISA para o IBM PC.

2) a. Seja *IPC* o número médio de ciclos de clock por instrução, a fórmula do desempenho de um programa é dada por:

$$Tempo\ de\ Execução = Nº\ de\ instruções * CPI * tempo\ do\ ciclo$$

b. A estratégia CISC tem como objetivo principal completar uma tarefa com o menor número de instruções possíveis. Isso é possível através da construção de hardwares capazes de identificar e executar instruções complexas.

Já na estratégia RISC, as instruções são as mais modulares possíveis, capazes de serem executadas em um ciclo de clock. Instruções complexas são divididas em instruções menores.

Enquanto a estratégia CISC visa reduzir a variável *Nº de instruções* ao preço do *CPI*, a abordagem RISC faz o inverso, o *CPI* diminui e o *Nº de instruções* cresce.

c. IPC = 1.2
CPI = 0.83333

3) a. A principal vantagem da arquitetura de pilha é a sua simplicidade. O código objeto gerado é compacto e os compiladores para esse tipo de máquina são mais simples e rápidos de serem construídos do que o de outras máquinas. Um exemplo de máquina de pilha é a calculadora polonesa inversa.

b. A principal característica da arquitetura de registradores é a divisão de instruções em 2 categorias: acesso à memória (load e store entre memória e registradores) e operações ALU (apenas entre registradores). Um exemplo dessa arquitetura foi o computador mainframe CDC 6600. Essa arquitetura também está presente em processadores vetoriais (incluindo também GPUs).

c. A principal diferença entre essas arquiteturas é a presença de um registrador acumulador, responsável pelo armazenamento de resultados intermediários de operações lógicas. Na arquitetura de pilha, a falta do acumulador é compensada com os operandos sempre implícitos no topo da pilha.

As duas arquiteturas são similares no ponto

4) a. As principais limitações estão relacionadas à constituição física dos processadores. Uma dessas limitações é o aquecimento. Quando um transistor muda de estado, é gerada eletricidade, que por sua vez, gera calor. Quanto maior a frequência do processador, mais frequentes são as mudanças de estados e a geração de calor.

b. $Cycle\ Length = \frac{1}{f} = \frac{1}{400000000} = 2,5ns$

5) a. Para números reais, o mais comum é o padrão IEEE 754 para representação em ponto flutuante. Nessa representação, o bit mais significativo é o sinal, com 0 para números positivos e 1 representando sinal negativo. Os 8 bits seguintes representam o expoente. Os 23 bits menos significativos representam a mantissa.

Exemplo: Dada a representação simples 0 10000000 110 0000 0000 0000 0000 0000.

- O bit mais significativo é 0, portanto o valor é positivo
- Em decimal, os 8 bits seguintes equivalem a 128. Logo, o valor do expoente será 128 - 127 = 1
- Por fim, temos 1,11 em binário (o primeiro 1 é implícito por estar normalizado). Em decimal, esse valor é 1,75.

Arranjando esses resultados, o valor representado em ponto flutuante equivale a 1,75 * 2^1 = 3,5

b. Para inteiros, o padrão usado depende se o sinal é considerado ou não. Para inteiros sem sinal, a representação é dada por binário básico. Para realizar a conversão para decimal, o processo é somar cada dígito do número multiplicado por 2 elevado a sua posição, indo de 0 a n-1 (sendo n o número total de dígitos).

Exemplo: Em 16 bits, 1001 0011 0100 1011

- $2^{15} + 2^{12} + 2^9 + 2^8 + 2^6 + 2^3 + 2^1 + 2^0 = 37707$

Na soma feita acima, os 1s e 0s foram deixados implícitos.

Já para inteiros com sinal, o padrão usado é o complemento a 2. Nesse modelo de representação, o bit mais significativo corresponde ao sinal do número. Enquantos os n-1 bits seguintes representam a magnitude do número.

Exemplo: Em 16 bits, 1001 0011 0100 1011 (mesmo valor do exemplo anterior)

- $-2^{15} + 2^{12} + 2^9 + 2^8 + 2^6 + 2^3 + 2^1 + 2^0 = -27829$

6) a. Os dois tipos de ordenações são chamadas *big-endian* e *little-endian*. Na ordenação *big-endian*, os dados são armazenados de forma que o byte mais significativo fica no endereço

menor e o resto da palavra, em endereços crescentes. Já na ordenação *little-endian*, é o byte menos significativo que fica no endereço menor, enquanto o resto é colocado em endereços em ordem crescente.

Essa arrumação ocorre em dados imediatamente maiores que 1 byte, por exemplo, dados do tipo *short* em C, que têm tamanho de 2 bytes.

b. A arquitetura 8051 (ou MCS-51) da Intel é um exemplo de arquitetura puramente *little-endian*. Como exemplo de arquitetura *big-endian*, temos a série 68000 da Motorola, produzida de 1979 até 1994.

c. Hoje, muitas arquiteturas permitem configurar a ordenação. A família ARM é um exemplo, originalmente ela era apenas *little-endian*, mas hoje é configurável.

d. Diretamente, não. A troca de dados entre computadores está sujeito a um protocolo de comunicação, uma série de regras que dita, inclusive, como os dados devem ser ordenados. O mais comum em protocolos de rede é a ordenação *big-endian*.

e.

1000	**1001**	**1002**	**1003**
1101 1100	0001 0010	1001 0000	1011 1010

7) a.

- Indireto via memória
  LDA @enderreço
  O valor contido na posição @endereço é tratado como um endereço, o qual é acessado e tem seu valor carregado no acumulador.

b. No modo indexado, são somados os 2 valores localizados na memória e o resultado é um endereço na memória, que contém um valor de interesse. No modo deslocamento, a soma é feita entre um valor na memória e um imediato, o resultado também é um endereço a ser acessado na memória.

c.

d. No modo indireto, é dado um endereço de memória, que contém o operando. No modo imediato, o próprio operando é passado.

8) O código hexadecimal resultando da instrução é 0x39C8.

00111001
11001000

9) a. Direto

b. Relativo ao PC

c. Direto e Indireto

d. Direto, Indireto e Imediato

e. Direto, Indireto e Imediato

f. Direto, Indireto e Imediato

g. Direto, Indireto e Imediato

h. Acumulador
i. Direto e Indireto
j. Direto

10)

```
;-------------------------------------------------
; Programa: EX 10 - LISTA 1
; Autor: GABRIEL AUREO DE OLIVEIRA CAMPOS
; Data:30/08/2018
;-------------------------------------------------
org 100
  pares: dw 0
  impares: dw 0
  i: db 0
  pt: dw x
  x: dw 2,3,4,5,6,7,8,9,10,11

org 0
  sta i
  jmp LOOP

LOOPCHK:
  lda i
  add #1
  sta i
  lda #11
  sub i
  jn LOOPEND

LOOP:
;calcula o endereço de x[i]
  lda i
  shl
  add #x
  sta pt
;verifica se é par
  lda @pt
  and #1
  jnz IMPAR
  lda pares
  add @pt
  sta pares
  jmp LOOPREPEAT
```

```
IMPAR:
  lda impares
  add @pt
  sta impares

LOOPREPEAT:
  jmp LOOPCHK

LOOPEND:
  hlt
```

11) **PC**: 0009 **A**: 68h **B**: 64h **Y**: 14h **ACC:** 9Bh

Y é inicializado com 14h (20 em decimal) e esse valor não é alterado.

A é inicializado com o valor de Y em modo de endereçamento direto, portanto, o que é armazenado é o endereço de Y, 68h.

B é inicializado com valor 2. Porém, durante a execução, o valor 80 é carregado no acumulador. Logo depois, o comando **ADD** @A acessa o valor contido em A como um endereço na memória. Como A contém o endereço de Y, o efeito é adicionar o valor de Y ao acumulador, resultando em 64h. Por fim, o comando **STA** B altera o valor de B de 2 para 64h.

O comando NOT realiza uma operação unária de negação sobre o acumulador, cujo valor é 64h (0b01100100) no momento. O resultado é 9Bh (0b10011011)

Analisando as instruções do programa:

- **LDA** #80 22 50 2 BYTES
- **ADD** @A 31 64 00 3 BYTES
- **STA** B 10 66 00 3 BYTES
- **NOT** 60 1 BYTE
- **HLT** FC 1 BYTE

Como o PC parte de 0000 e o programa termina na instrução HLT, temos que a soma dos tamanhos das instruções resulta em 9.

12) a. **PC**: 0012

b.

```
0060: 0000 0000 1001 6002
0068: 1201 7002 1401 1002
0070: 2055 2444 4040 6633
0078: 2835 0000 0000 0000
```

c. **SP**: FFFF

d. **ACC**: 22