Lista de Exercícios

1) Considere o projeto de CPU **ProtoOne** v2.3 para os itens que se seguem:
   a) Altere o microcódigo para incluir uma instrução de subtração ao ISA sem alterar a ALU, discuta sobre as dificuldades encontradas e, se foi ou não fazer alguma alteração no circuito da CPU.
   b) Modifique a ALU da CPU original para incluir as operações incremento, utilizando apenas um único componente de soma já incluído no projeto.
   c) Encontre soluções para a execução de operações de subtração usando a CPU do item de exercício anterior (item 1.b). Utilize recursos por software (em *assembly*) e compare com uma solução por hardware (em microcódigo).
   d) Compare a eficiência (percentual) das 2 soluções do item anterior. Qual o motivo da diferença?

2) Considere o *hardware* da CPU **ProtoTwo** para os itens que se seguem:
   a) Proponha um conjunto de instruções (Assembly e ISA) com instruções de tamanho fixo de 16 *bits* que consiga resolver problemas de alto nível contendo estruturas como: IF-THEN-ELSE, FOR-NEXT, WHILE-DO, REPEAT-UNTIL, SWITCH-CASE, além de operação com vetores e chamada de funções.
   b) Escreva programas no *assembly* do item anterior para: multiplicação de matrizes 5x5, função fatorial (recursiva), resolução de equação de segundo grau.
   c) Escreva o microcódigo para 5 das instruções ISA propostas no item 2.a.

3) Considere um microcódigo que apresenta micro saltos conforme a tabela a seguir:

Micro salto	Valor	Descrição
Next	0000	Executa a microinstrução e incrementa uPC
Spin	0001	Executa a microinstrução, mas não modifica uPC
Fetch	0010	Executa a microinstrução e salta para a microinstrução do FETCH (uPC=0)
Dispatch	0011	Executa a microinstrução e salta para o endereço da primeira microinstrução correspondente à instrução ISA contida no IR
CondN	0100	Incrementa uPC e executa a microinstrução se FLAG N=1 (*negative*)
CondZ	0101	Incrementa uPC e executa a microinstrução se FLAG Z=1 (*zero*)
CondC	0110	Incrementa uPC e executa a microinstrução se FLAG C=1 (*carry*)
CondV	0110	Incrementa uPC e executa a microinstrução se FLAG V=1 (*overflow*)
CondLE	1100	Incrementa uPC e executa a microinstrução se FLAG N=1 ou Z=1 (*less than or equal*)
CondGT	1101	Incrementa uPC e executa a microinstrução se FLAG N=0 e Z=0 (*greater than*)

   a) Construa um circuito que gere os sinais Exec, JUMPsel.

4) Suponha um processador baseado em pilha que inclui as operações de pilha PUSH e POP. As operações aritméticas envolvem automaticamente um ou dois elementos do topo da pilha. Comece com uma pilha vazia. Quais elementos restam na pilha depois que as instruções a seguir são executadas?

   PUSH 4
   PUSH 7

PUSH 8
ADD
PUSH 10
SUB
MUL

5) Um processador de 16 *bits* (dados e endereços) foi utilizado para a construção de um computador com uma Arquitetura de E/S do tipo MMIO. Sabendo que para a seleção de endereços de entrada e saída foi utilizado uma porta AND com 8 entradas ligadas aos *bits* de endereço A15 a A8, pergunta-se:
   a) Quantos endereços de E/S foram disponibilizados, uma vez que se utilizou decodificação exaustiva?
   b) Projete um sistema capaz de expandir a capacidade de memória física para 256 KiB em "páginas" de 16 KiB, utilizando-se apenas um endereço de E/S para isso.

6) Leia o texto a seguir, a respeito do CHIP-8.

Em meados da década de 1970, foi criada por Joseph Weisbecker, uma linguagem de programação interpretada, inicialmente usada nos microcomputadores COSMAC VIP e Telmac 1800 de 8 *bits*, chamada CHIP-8. Essa linguagem foi escolhida para permitir que os videogames sejam mais facilmente programados para esses computadores. Aproximadamente quinze anos depois que o CHIP-8 foi introduzido, interpretadores derivados apareceram para alguns modelos de calculadoras gráficas (do final da década de 1980 em diante, esses dispositivos portáteis em muitos aspectos têm mais poder de computação do que a maioria dos microcomputadores de meados da década de 1970 para amadores).

Memória

O CHIP-8 foi implementado em sistemas 4K, ou seja, tinham 4096 (0x1000) locais de memória, todos de 8 *bits*. A maioria dos programas escritos para o sistema começa no local de memória 512 (0x200) e não acessa nenhuma memória abaixo do local 512 (0x000-0x200), mas em implementações modernas, esses *bytes* inferiores de memória, são usados para armazenar dados de fontes. Os 256 *bytes* superiores (0xF00-0xFFF) são reservados para atualização da tela, e os 96 *bytes* abaixo disso (0xEA0-0xEFF) foram reservados para a pilha de chamadas, uso interno e outras variáveis.

Registradores

O CHIP-8 tem 16 registradores de dados de 8 *bits* denominados V0 a VF. O registrador VF funciona como um sinalizador para algumas instruções; portanto, deve ser evitado para armazenamento. Em uma operação de adição, VF é o sinalizador de transporte, enquanto na subtração, é o sinalizador "não emprestar". Em instruções de desenho na placa gráfica, o VF é definido na colisão de pixels.
O registrador de endereço, que é denominado I, tem 12 *bits* de largura e é usado com vários *opcodes* que envolvem operações de memória.

A pilha

A pilha só é usada para armazenar endereços de retorno quando as sub-rotinas são chamadas. A versão original do RCA 1802 alocou 48 *bytes* para até 12 níveis de aninhamento; implementações modernas geralmente têm mais.

Descrição do ISA (simplificado, apenas parte das instruções)

Opcode	Pseudo C	Descrição
1NNN	`goto NNN`	Salta para o endereço NNN.
3XNN	`if (Vx == NN) inc PC`	Pula a próxima instrução se VX é igual a NN.
4XNN	`if (Vx != NN) inc PC`	Pula a próxima instrução se VX é diferente de NN.
5XY0	`if (Vx == Vy)`	Pula a próxima instrução se VX é igual a VY.
6XNN	`Vx = N`	Carrega VX com o valor NN.
7XNN	`Vx = Vx + N`	Soma NN a VX. (*Carry flag* não é alterado).
8XY0	`Vx = Vy`	Carrega VX com o valor de VY.
8XY1	`Vx = Vx or Vy`	Carrega VX com VX or VY.
8XY2	`Vx = Vx and Vy`	Carrega VX com VX and VY.
8XY3	`Vx = Vx xor Vy`	Carrega VX com VX xor VY.
8XY4	`Vx = Vx + Vy`	Soma VY a VX. VF é alterado para 1 quando há *carry*, e para 0 quando não há.
8XY5	`Vx = Vx - Vy`	Subtrai VY de VX. VF é carregado com 0 quando over um *borrow*, e 1 caso contrário.
8XY6	`Vx = Vx >> 1`	Armazena o *bit* menos significativo de VX em VF e então desloca VX para a direita em 1 *bit*.
8XY7	`Vx = Vy - Vx`	Subtrai VX de VY, armazenando em VX. VF é carregado com 0 quando over um *borrow*, e 1 caso contrário.
8XYE	`Vx = Vx << 1`	Armazena o *bit* mais significativo de VX em VF e então desloca VX para a esquerda em 1 *bit*.
9XY0	`if (Vx != Vy) inc PC`	Salta a próxima instrução se VX não é igual a VY.
FX1E	`I = I + Vx`	Adiciona VX a I. VF não é afetado.

a) Observe o conjunto ISA e proponha uma linguagem *Assembly* para o processador.
b) Faça um projeto do circuito da CPU CHIP-8 para a construção de uma versão física (*hardware*) da máquina virtual, utilizando um modelo CISC. Considere que já existam os blocos funcionais: ALU, somadores, multiplexadores, Unidade de Controle, incrementadores e registradores (conjunto de FF-D).
c) Especifique cada sinal gerado pela unidade de controle e a sua finalidade. Indique a quantidade de *bits* em cada linha indicada no diagrama.
d) Identifique os modos de endereçamento utilizados no conjunto de instruções apresentado, mostrando exemplos cara cada modo de endereçamento.
e) Sabendo que o controlador da máquina é implementado em uma memória ROM, utilizando, descreva como será realizada a instrução que "soma VY a VX. VF é alterado para 1 quando há *carry*, e para 0 quando não há" (código hexadecimal 8XY3), mostrando cada passo do microprograma e a forma de implementação na ROM. Não esqueça da busca da instrução em memória.

f) Considerando uma organização estruturada de computadores, conforme estudado, "desenhe" a estrutura multinível (hierarquia de máquinas) de uma máquina moderna executando um programa escrito para CHIP-8. Identifique quais máquinas são virtuais, quais reais, como são implementados cada um desses níveis (compilados, interpretados...).
g) Faça um esboço de um circuito para a ULA desta máquina ISA, usando o Logisim.
h) Uma das instruções do CHIP-8 (0xFX33 com mnemônico DEQ, VX), armazena a representação BCD (binary-coded decimal) do valor armazenado em no registrador VX, com o mais significativo dos 3 dígitos no endereço I (centenas), o dígito do meio em I + 1 (dezenas), e o dígito menos significativo em I + 2 (unidades).
   i) Construa um programa em assembly MIPS no MARS, que realize esta mesma operação, considerando as variáveis tipo byte chamada V e tipo word chamada E, como parâmetros de entrada para realização do procedimento, armazenando o conteúdo de V, convertido em BCD, nos endereços E e subsequentes.
   ii) Suponha que o endereço armazenado em E seja sempre um múltiplo de 4.

7) Construa um programa em assembly MIPS no MARS, que leia um número inteiro entre 0 e 255 digitado pelo usuário (*syscall 1*) e imprima uma mensagem para indicar se o mesmo apresenta um número par ou ímpar de 1s na sua forma binária. O programa deve apresentar uma mensagem de erro caso o valor esteja fora da faixa (de 0 a 255).
Dica: tente pensar em deslocamentos sucessivos do número...

8) Leia as informações a seguir sobre o processador Z80:

O processador **Z80**, também conhecido como **Zilog Z80** é um microprocessador de 8 *bits*, projetado e vendido a partir de 1976, a princípio foi utilizado em computadores *desktop* e em sistemas embarcados. Junto com a **MOS Technology**, criadora do microprocessador **6502**, dominou o mercado de computadores de 8 *bits* final da década de 1970 e em toda década de 1980. A *Intel* tentou superar o Z80 e lançou uma versão melhorada do 8085 (microprocessador Intel de 8 *bits*), mas acabou desistindo do projeto pela superioridade do Z80.

O processador possui registradores de propósito geral de 8 *bits* denominados B, C, D, E, H, e L, além do acumulador (registrador A) e de um registrador de *Flags*, chamado F. Esses registradores também podem ser usados aos pares (AF, BC, DE e HL) para algumas operações. Além dos registradores de propósito geral, o Z80 também apresentava registradores de propósito especial, como o PC, SP, IX e IY (estes 2 últimos eram usados como índices e operações muito poderosas), todos de 16 *bits*. As instruções do Z80 apresentam o tamanho de 1 a 4 *bytes*, entre estão as apresentadas a seguir.

Mnemônico	Código ISA	Efeito
LD A, B	0x78	A ← B
LD A, C	0x79	A ← C
LD A, D	0x7A	A ← D
LD B, A	0x47	B ← A
LD B, C	0x41	B ← C
LD B, H	0x44	B ← H
LD B, (HL)	0x70	B ← valor da posição de memória apontada por HL
LD (HL), B	0x46	valor da posição de memória apontada por HL ← B
LD (HL), A	0x47	valor da posição de memória apontada por HL ← A
ADD B	0x80	A ← A + B

ADC C	0x89	A ← A + C + *carry*
SUB D	0x92	A ← A - D
SBC E	0x9B	A ← A - E - *carry*
AND H	0xA4	A ← A AND H
XOR L	0xAD	A ← A XOR L
OR (HL)	0xB6	A ← A OR memória na posição apontada por HL
CP A	0xBF	compara A com A (subtrai, seta *flags*, não armazena o resultado)
INC B	0x04	B ← B + 1
INC C	0x14	C ← C + 1
DEC B	0x05	B ← B - 1
DEC C	0x15	C ← C - 1
LD B, ii	0X06 0xii	B ← valor do próximo *byte*
LD C, ii	0X0E 0xii	C ← valor do próximo *byte*
LD D, ii	0X16 0xii	D ← valor do próximo *byte*
LD E, ii	0X1E 0xii	E ← valor do próximo *byte*
RLC A	0xCB 0x07	rotaciona A para esquerda e armazena b7 no *carry*
RRC A	0xCB 0x0F	rotaciona A para direita e armazena b0 no *carry*
RL A	0xCB 0x17	rotaciona A e carry (9 *bits*) para esquerda
RR A	0xCB 0x1F	rotaciona A e carry (9 *bits*) para direita
SLL B	0xCB 0x20	desloca B um *bit* para a esquerda (lógica) e armazena b7 no *carry*
SRL B	0xCB 0x28	desloca B um *bit* para a direita (lógica) e armazena b0 no *carry*
SLA B	0xCB 0x30	desloca B um *bit* para a esquerda (aritmética) e armazena b7 no *carry*
SRA B	0xCB 0x38	desloca B um *bit* para a direita (aritmética) e armazena b0 no *carry*
BIT 0, B	0xCB 0x40	testa o *bit* 0 do registrador B, alterando o *flag* Z
BIT 0, C	0xCB 0x41	testa o *bit* 0 do registrador C, alterando o *flag* Z
BIT 1, C	0xCB 0x49	testa o *bit* 1 do registrador B, alterando o *flag* Z
RES 7, B	0xCB 0xB8	reseta o *bit* 7 do registrador B
SET 7, B	0xCB 0xF8	seta o *bit* 7 do registrador B
LDI	0xED 0xA0	um *byte* de dados é transferido da posição de memória apontada HL para a posição apontada por DE; em seguida, ambos os pares de registradores são incrementados e o par de registradores do contador de *bytes* (BC) é decrementado
LDIR	0xED 0xB0	executa o mesmo que **LDI**, porém repete até que BC seja zero
LDD	0xED 0xA8	similar a **LDI**, porém HL e DE sãos decrementados
LDDR	0xED 0xB8	similar a **LDIR**, com incremento de HL e DE
LD (IX+ii), B	0xDD 0x70 0xii	Carrega a posição de memória apontada por IX + ii com o registrador B
LD (IX+ii), C	0xDD 0x71 0xii	Carrega a posição de memória apontada por IX + ii com o registrador C
LD (IX+ii), A	0xDD 0x77 0xii	Carrega a posição de memória apontada por IX + ii com o registrador A

a) Observando pequeno subconjunto de instruções apresentado, podemos afirmar que o processador é construído usando o paradigma CISC ou RISC? Justifique a resposta.
b) Projete uma Unidade de Lógico Aritmética para a execução das instruções apresentadas neste subconjunto de instruções.
c) O processador faz uso de expansão de códigos, isto é, alguns valores de opcode são usados para indicar que o *opcode* está em outra região da instrução. Mostre como isso é feito com exemplos.
d) Mostre como são codificadas as instruções no ISA deste microprocessador (agrupe as instruções por tipos).
e) Quais os tipos de endereçamento estão descritos no subconjunto apresentado (inclusive os implícitos)? Apresente exemplos de cada.