1)

a) 32 bits, e são chamados de palavra (word).
b) 8 bits (1 byte), cada palavra ocupa 4 bytes.
c) 32 registradores.
d) Existem três formatos de instrução:
   Formato R: op (6 bits), rs (5 bits), rt (5 bits), rd (5 bits), shamt (5 bits), funct (6 bits), nessa ordem.
   Formato I: op (6 bits), rs (5 bits), rt (5 bits), constant (16 bits), nessa ordem.
   Formato J: op (6 bits), address (26 bits), nessa ordem.
   Todos eles somam 32 bits.
   op = opcode, rs = 1º registrador source, rt = 2º registrador source, rd = registrador destino, shamt = shift amount, funct = função de extensão.

2) Monociclo:
   Vantagens:
   - Mais simples de implementar;
   Desvantagens:
   - Ineficiente, pois os ciclo de clock é determinado pelo maior path;
   - Hardware duplicado, e então é mais caro;
   Multiciclo:
   Vantagens:
   - Eficiente, pois o ciclo de clock varia de acordo com a instrução;
   - Não tem hardware duplicado, sendo também mais barato;
   Desvantagens:
   - Mais difícil de implementar;
   (As vantagens e desvantagens são em relação a monociclo e multiciclo)

3) Vantagens:
   - Facilidade de implementação;
   - Custo de implementação;
   Desvantagens:
   - Essa limitação pode fazer com que o código fique complexo;

4) Pois as operações lógicas e aritméticas só são permitidas entre registradores, tudo que envolve memória deve ser através do load e store.
   Vantagens:
   - Como os registradores são mais rápidos que a memória, isso faz com que as operações sejam mais rápidas;
   Desvantagens:
   - Sobrecarga de leitura e escrita na memória;

5)

a) $Desempenho_x$ / $Desempenho_y$ = $Tempo\ de\ Execução_y$ / $Tempo\ de\ Execuçaõ_x$
Tempo de Execução (CPU) = Ciclos de Clock / Frequência
Ciclos de Clock = Número de Instruções * CPI
Considerando uma instrução: Ciclos de Clock = CPI
Tempo de Execução:
P1: 1.5/2GHz (0.75)
P2: 1.0/1.5GHz (0.66)
P3: 2.5/3GHz (0.83)
O **processador 2** tem um melhor desempenho.

b) Tempo de CPU = Número de Instruções * CPI / Frequência
Número de Instruções = (Tempo de CPU * Frequência) / CPI
Ciclos = Instruções * CPI
P1: 1.3*10^10 instruções, 2.0*10^10 ciclos;
P2: 1.5*10^10 instruções, 1.5*10^10 ciclos;
P3: 1.2*10^10 instruções, 3.0*10^10 ciclos.

c) **P1**: 3.34GHz;
P2: 2.57GHz;
P3: 5.14GHz.

6)

a) Tempo de Execução (CPU) = Ciclos de Clock / Frequência.
A **segunda** implementação (Com o que foi calculado abaixo:
Tempo de P1: 1.86*10^-3
Tempo de P2: 1.0*10^-3).

b) CPI = sum 1 to n (CPI_i * $Número\ de\ instruções_i$/Número de instruções)
P1: 2.8;
P2: 2.0.

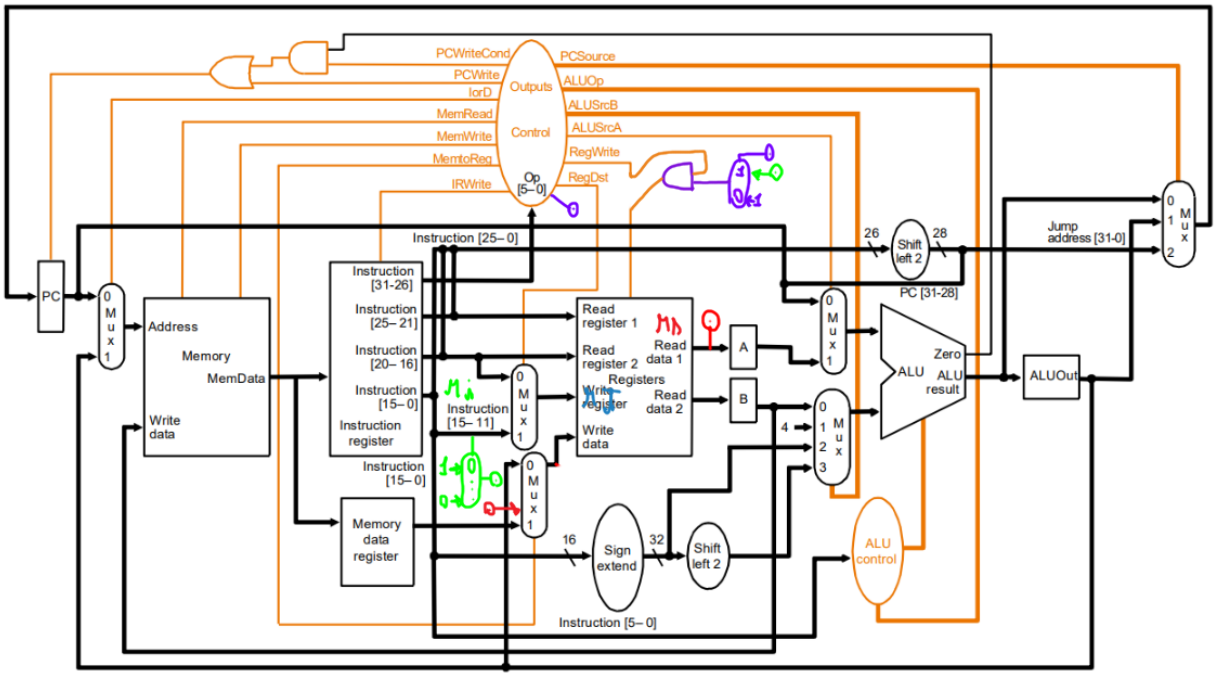
c) Ciclos de clock = Números de Instruções * CPI
P1: 2.8*10^6;
P2: 2.0*10^6.

7) Deve ser adicionado um novo sinal de controle, mais precisamente, uma extensão do ALUSrcB. Como já estão sendo utilizados os 4 (2 bits) sinais, devemos extender para 8 (3 bits) (sim, existem outras formas de fazer isso). Assim, teremos o sinal 4 (100). A ligação de ALUSrcB com o MUX permanece a mesma, a única diferença é quando for o sinal 4, a saída do MUX deve ser 0. Dessa forma, adicionaremos 0 ao 1º Operando da ALU. Obviamente, isso acarreta numa leve mudança na máquina de estados.
Vai ser necessário alterar o Control para quando esse comando ocorrer:
ALUSrcA seja 1 (1º Operando da ALU vem de B);
ALUSrcB seja 4 (2º Operando da ALU é 0);

ALUOP seja 00 (ALU faz soma);
E no próximo estado:
MemToReg seja 0 (Valor escrito na entrada “Write data” dos registradores vem da ALUOut);
RegDst seja 0 (Número do registrador destino para escrita vem de rt);
RegWrite seja 1 (Registrador da entrada “Write register” recebe valor da entrada “Write data”);
Teria o mesmo CPI de um add (4).

8)

a) Sinal roxo: 1 se for cmove, 0 se não for cmove.
MUX roxo: 1 se receber 0, sinal verde se receber 1.
AND roxo: Para manter esse sinal funcionando normalmente e aplicar a lógica de salvar ou não rs em rt se ri for 0.
MUX verde: 1 se receber 0, 0 se receber qualquer outra coisa (como são 5 bits, não são tantas possibilidades).
Fio verde: Saindo de ri para o MUX verde, e a saída de MUX verde para o MUX roxo.
Fio vermelho: Será necessário aumentar a quantidade de sinais de MemToReg, para quando for cmove, ele ser 2 e essa conexão direta de rs com o MUX funcionar.



Novamente, uma alteração do Control será imprescindível:
MemToReg seja 2 (Valor escrito na entrada “Write data” dos registradores vem do fio vermelho;
RegDst seja 0 (Número do registrador destino para escrita vem de rt);
RegWrite seja 1 (Registrador da entrada “Write register” recebe valor da entrada “Write data”);
Sinal Roxo seja 1 (Fio verde passará para o AND);

Teria o mesmo CPI de um add (4).

b) Melhoraria o desempenho da máquina. Porque o CPI é menor; antes é necessário fazer um desvio condicional (4) e depois um move (4), totalizando um CPI de 8, mas, se o cmove tem o mesmo CPI do move, teremos apenas 4 no lugar de 8.