Questão 1

O programa calcula a divisão de dois números de 32 bits baseado no segundo algoritmo de divisão passado em aula e imprime ao final o resultado inteiro da solução e o resto.

Primeiramente, foi necessário a criação de uma função para realizar shift left em dois registradores, pelo fato de que o algoritmo utiliza o dobro de bits para o resto e cada registrador suporta somente 32 bits.

Explicação da função shift_left:

> O bit mais significativo da primeira parte do resto é salvo no registrador $t2. Posteriormente é usado a função sll(shift logical left) nos dois registradores que armazenam cada parte do resto, respectivamente de menor para maior, $t1 e $t0. Por fim, é somado o valor armazenado em $t2 a maior parte do resto $t0.

A função será referenciada como “shift_left” durante a explicação do código.

Inicialmente, carregamos as variáveis:

- $s0 -> irá conter a parte mais significativa do resto e que no final representará o resto da divisão
- $s1 -> irá conter a parte menos significativa do resto e que no final representará o resultado inteiro da divisão
- $s3 -> representa o divisor

A seguir é chamada a função shift_left passando como parâmetro os valores de $s0 e $s1 completando o primeiro passo do algoritmo.

Agora temos um loop com 32 interações, que representa a quantidade de bits dos operandos, a cada interação é verificado se $s0, parte mais significativa do resto, é menor que o divisor, $s3, se for pulamos para a operação resto_menor caso contrário a execução segue para o resto_maior.

resto_menor:

> Caso o resto seja menor que o divisor é chamado shift_left com $s0 e $s1 como parâmetros.

resto_maior:

> Caso o resto seja maior que o divisor é subtraído o divisor, $s3, da maior parte do resto, $s0. Posteriormente é dado um shift_left no resto com $s0 e $s1 como parâmetros. Por fim, é adicionado 1 a parte menos significativa do resto, $s1.

Depois da execução do loop o último passo do algoritmo é dar um srl(shift right logical) no $s0, parte mais significativa do resto que vem a ser o próprio resto. Ao finalizar essas operações, os registradores $s1 e $s0 guardam respectivamente a parte inteira e o resto da divisão e são impressos no terminal do software mars.

Questão 2

O programa calcula uma aproximação do cosseno de x usando a série de Taylor. Ele itera sobre uma sequência, calculando fatoriais e potências de acordo com a série de Taylor. O sinal do termo na série é alternado dependendo se a iteração é par ou ímpar. O loop é executado até atingir a condição de parada definida em $a1 (7). O resultado é armazenado em $f22 e depois transferido para $f12 para ser impresso no terminal. O programa é encerrado com uma chamada de sistema (syscall).

Primeiramente carregamos as variáveis de ponto flutuante nos respectivos registradores de iniciais “f” com o uso da instrução “ldc1”. Em seguida, transformamos o número de graus para radianos multiplicando-o por pi e dividindo-o por 180.

Após isso, são carregadas algumas variáveis úteis para o funcionamento do programa:

- $f22 -> irá conter o valor do cosseno
- $a0 -> contador do número de iterações do loop principal
- $a1 -> contém a condição de parada do loop (7 iterações, como solicitado no comando do exercício)
- $a2 -> variável inicializada com 1, varia de 1 para 0 no decorrer do programa para identificar se a interação é par ou ímpar e assim achar o sinal do termo da série
- $f24 -> contém o número 2, usado para incrementar o valor de n

A seguir, começa o loop principal do programa. Algumas variáveis precisam ser inicializadas novamente a cada interação:

- $f12 -> armazena o valor do fatorial
- $f14 -> contador do loop da função “calcula_fatorial”
- $f16 -> armazena o número 1 para ser incrementado ao contador
- $f18 -> armazena o valor da potenciação
- $f20 -> contador do loop da função “calcula_potenciacao”

Depois, o programa vai para a função “calcula_fatorial” e armazena o valor do fatorial de n em $f12.

calcula_fatorial:

Esta função consiste em um loop que a cada interação multiplica o valor de $f12, que começa com 1, pelo valor de $f14, contador que aumenta em uma unidade a cada ciclo até ser igual a n. Assim, acha-se o fatorial de n e este fica armazenado em $f12.

Em seguida, o programa é direcionado para a função calcula_potenciacao.

calcula_potenciacao:

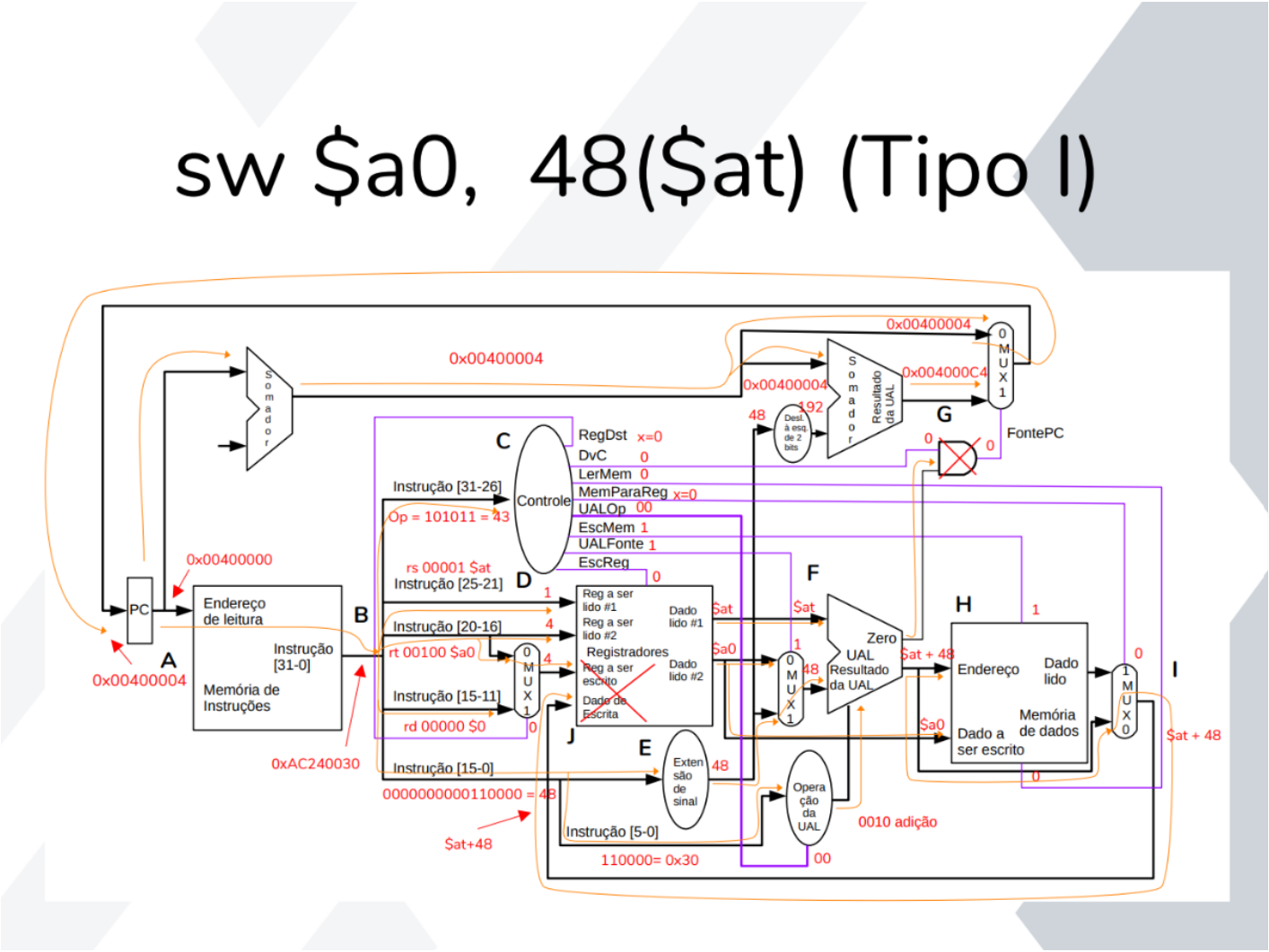
Nesta função, $f18 (inicializado com 1) é multiplicado a cada interação por $f10, que armazena o valor de x. Um contador é incrementado a cada ciclo do loop até ser igual a n. Assim, acha-se o valor de x elevado à enésima potência e esse valor fica armazenado em $f18.

Após achar os valores do fatorial e da potenciação da respetiva interação, divide-se $f18 por $f12 (potenciação por fatorial) para achar o termo da série. A seguir, verifica-se o valor do registrador $a2. Se for 1, o termo será negativo e, portanto, será subtraído de $f22, que contém o valor atual da série. Logo depois, o registrador $a2 é atualizado com novo valor, neste caso, 0. Por outro lado, se o registrador $a2 for 0, o termo da série será positivo e somado a $f22. Seguidamente, o valor de $a2 será atualizado para 1.

Posteriormente, o programa verifica se ainda não foi atingida a condição de parada comparando o contador do loop, $a0, com $a1, que armazena o número 7. Caso $a0 seja menor que $a1, todo o processo explicado anteriormente é repetido novamente, caso contrário, o loop é encerrado e o valor da série, representando o cosseno de x, contido em $f22, é impresso no terminal mediante a função imprime_cosseno. Em seguida, o programa é encerrado a partir de uma chamada de sistema (syscall).

Questão 5

A seguir descrevemos o fluxo dos sinais no processador monociclo, na execução da instrução sw $a0, 48($at):



(A) Na borda de subida do sinal de relógio, o registrador PC é carregado com o endereço da próxima instrução que será executada pelo processador monociclo. Este endereço vai até o barramento de endereços da memória de instruções e a entrada de um somador.

(B) Na saída da memória de instruções temos a instrução 0xAC240030. Esta é a instrução sw $a0, 48($at). Os vários campos desta instrução são separados: o campo imediato da instrução jump (instrução[25-0]), opcode (instrução[31-26]), registrador rs (instrução[25-21]), registrador rt (instrução[20-16]), registrador rd (instrução[15-11]) e campo imediato (instrução[15-0]).

(C) O campo opcode (instrução[31-26]) é decodificado pela unidade de controle. Sinais de controle são gerados. Estes sinais estão apresentados na tabela 1: RegDst = 0, UALFonte = 1, MemParaReg = X = 0, EscReg = 0, LerMem = 0, EscMem = 1, DvC = 0, UALOp = 00.

(D) No banco de registradores chegam os endereços dos registradores rs (instrução[25- 21]) = $at e rt (instrução[20-16]) = $a0, na entrada dos endereços dos registradores que serão lidos. O conteúdo destes registradores aparecem nas saídas do banco de registradores “dado lido #1” e “dado lido #2”. O banco de registradores recebe o sinal de controle EscReg = 0. Neste problema, o endereço do registrador é a entrada 0 do MUX. Nesta entrada temos o registrador $a0. O sinal de controle RegDst = 0 seleciona esta entrada deste MUX.

(E) Ocorre a extensão do sinal no campo imediato (instrução[15-0]) =48. Este sinal passa de 16 para 32 bits. O campo funct(instrução[5-0]) = 0x30 é extraído do campo imediato e ligado a entrada do circuito que gera o código de operação da UAL. Este circuito também recebe o sinal de controle UALOp = 00. Usando a tabela 2, vemos que é gerado o sinal de controle para a UAL 0010, solicitando uma adição para a ULA.

(F) Em uma das entradas da ULA temos um MUX que recebe o sinal de controle UALFonte = 1. É selecionado o valor imediato estendido 48 para uma das entradas da ULA. Esta recebe em sua outra entrada o conteúdo do registrador rs = $at. O sinal de controle 0010 do controle de operação da ULA faz com que as entradas sejam somadas. Na saída da ULA temos o valor $at+48.

(G) Um segundo somador calcula o endereço de desvio de instruções condicionais beq. O valor imediato = 48 é deslocado por 2 bits para a esquerda, o que equivale a uma multiplicação por 4. O valor imediato multiplicado por 4, 192, é uma das entradas do somador. A outra entrada é o endereço da próxima instrução que será executada, PC+4 = 0x0040 0004. Na saída do somador temos o endereço 0x0040 00C4. Este é o endereço de desvio de uma instrução beq. Na saída do somador temos um multiplexador com um sinal de controle FontePC. Uma porta AND gera o sinal FontePC, que é igual a 0, porque uma de suas entradas tem o sinal de controle DvC igual a 0. Neste caso, a saída da porta, o sinal FontePC será sempre 0, independente do valor do sinal zero vindo da ULA. Com FontePC = 0 selecionamos o endereço PC+4 = 0x0040 0004. Este endereço é a entrada 0 do MUX com o sinal de controle jump. Como este sinal de controle é 0, a saída deste multiplexador será o valor de PC+4, o endereço da próxima instrução e não o endereço de desvio incondicional. O endereço de PC+4 vai para a entrada do registrador PC.

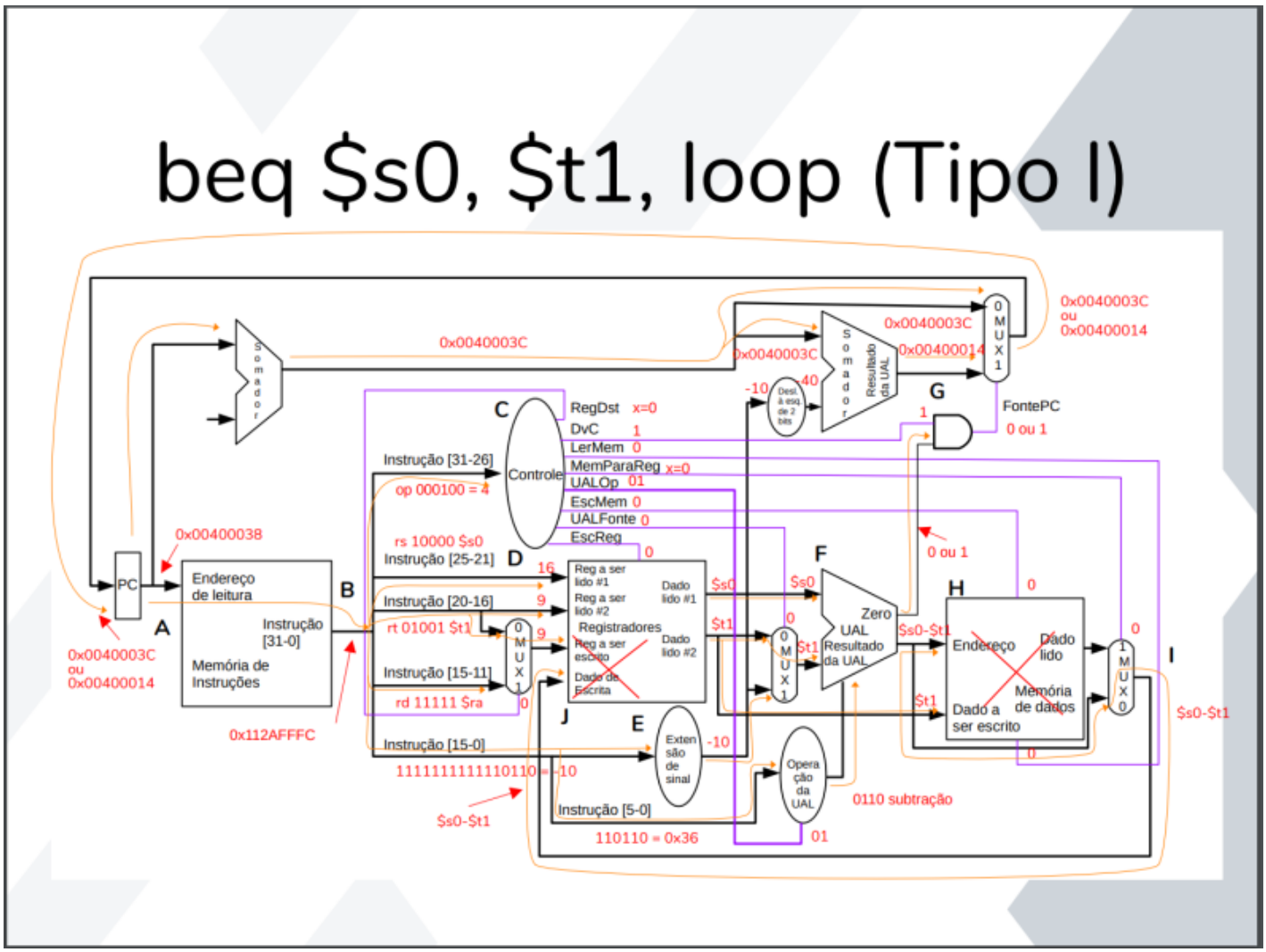
(H) A memória de dados recebe a soma $at+48 no barramento de endereços e o conteúdo do registrador $a0 na entrada “dado a ser escrito”. Esta memória recebe os sinais de controle EscMem = 1 e LerMem = 0. Ocorre a escrita na memória de dados. O conteúdo do endereço de $at+48 aparece na saída “dado lido” da memória de dados.

(I) O multiplexador com o sinal MemParaReg = X = 0 tem na sua saída o resultado do somador $at+48. Este sinal é levado até a entrada “dado de escrita” do banco de registradores.

(J) O sinal EscReg = 0 do banco de registradores não permite a escrita neste banco. Nesta borda de subida o registrador PC também é atualizado. Gravamos neste registrador o endereço da próxima instrução que será executada pelo processador, o endereço 0x0040 0020.

Questão 6

A seguir descrevemos o fluxo dos sinais no processador monociclo, na execução da instrução beq $s0, $t1, loop:



(A) Na borda de subida do sinal de relógio, o registrador PC é carregado com o endereço da próxima instrução que será executada pelo processador monociclo. Neste problema, PC é carregado com o endereço 0x0040 0038. Este endereço vai até o barramento de endereços da memória de instruções e a entrada de um somador.

(B) Na saída da memória de instruções temos a instrução 0x112AFFFC. Esta é a instrução beq $s0, $t1, loop. Os vários campos desta instrução são separados: o campo imediato da instrução jump (instrução[25-0]), opcode (instrução[31-26]), registrador rs (instrução[25-21]), registrador rt (instrução[20-16]), registrador rd (instrução[15-11]) e campo imediato (instrução[15-0]).

(C) O campo opcode (instrução[31-26]) é decodificado pela unidade de controle. Sinais de controle são gerados. Estes sinais estão apresentados na tabela 1: RegDst = X = 0, Jump = 0, UALFonte = 0, MemParaReg = X = 0, EscReg = 0, LerMem = 0, EscMem = 0, DvC = 1, UALOp = 01.

(D) No banco de registradores chegam os endereços dos registradores rs (instrução[25- 21]) = $s0 e rt (instrução[20-16]) = $t1, na entrada dos endereços dos registradores que serão lidos. O conteúdo destes registradores aparecem nas saídas do banco de registradores “dado lido #1” e “dado lido #2”. O banco de registradores recebe o sinal de controle EscReg = 0. Neste problema, o endereço do registrador é a entrada 0 do MUX. Nesta entrada temos o registrador $t1. O sinal de controle RegDst = 0 seleciona esta entrada deste MUX.

(E) Ocorre a extensão do sinal no campo imediato (instrução[15-0]) = -10. Este sinal passa de 16 para 32 bits. O campo funct(instrução[5-0]) = 0x36 é extraído do campo imediato e ligado a entrada do circuito que gera o código de operação da UAL. Este circuito também recebe o sinal de controle UALOp = 01. Usando a tabela 2, vemos que é gerado o sinal de controle para a UAL 0001, solicitando uma subtração para a UAL.

(F) Em uma das entradas da ULA temos um MUX que recebe o sinal de controle UALFonte = 0. É selecionado o “dado lido 2”, que é o registrador rt = $t1, para uma das entradas da ULA. Esta recebe em sua outra entrada o conteúdo do registrador rs = $s0. O sinal de controle 0110 do controle de operação da ULA faz com que as entradas sejam subtraídas. Na saída da ULA temos o valor $s0-$t1.

(G) Um segundo somador calcula o endereço de desvio de instruções condicionais beq. O valor imediato = -10 é deslocado por 2 bits para a esquerda, o que equivale a uma multiplicação por 4. O valor imediato multiplicado por 4, -40, é uma das entradas do somador. A outra entrada é o endereço da próxima instrução que será executada, PC+4 = 0x0040 003C. Na saída do somador temos o endereço 0x0040 0014. Este é o endereço de desvio de uma instrução beq. Na saída do somador temos um multiplexador com um sinal de controle FontePC. Uma porta AND gera o sinal FontePC, que pode ser igual a 0 ou a 1, porque uma de suas entradas tem o sinal de controle DvC igual a 1 e a outra entrada pode ser 0 ou 1. Neste caso, a saída da porta, o sinal FontePC pode variar entre 0 ou 1, dependendo do valor do sinal zero vindo da ULA. Em caso de FontePC = 0 selecionamos o endereço PC+4 = 0x0040 003C, caso contrário selecionamos o endereço de desvio 0x00400014. O endereço que for selecionado vai para a entrada do registrador PC.

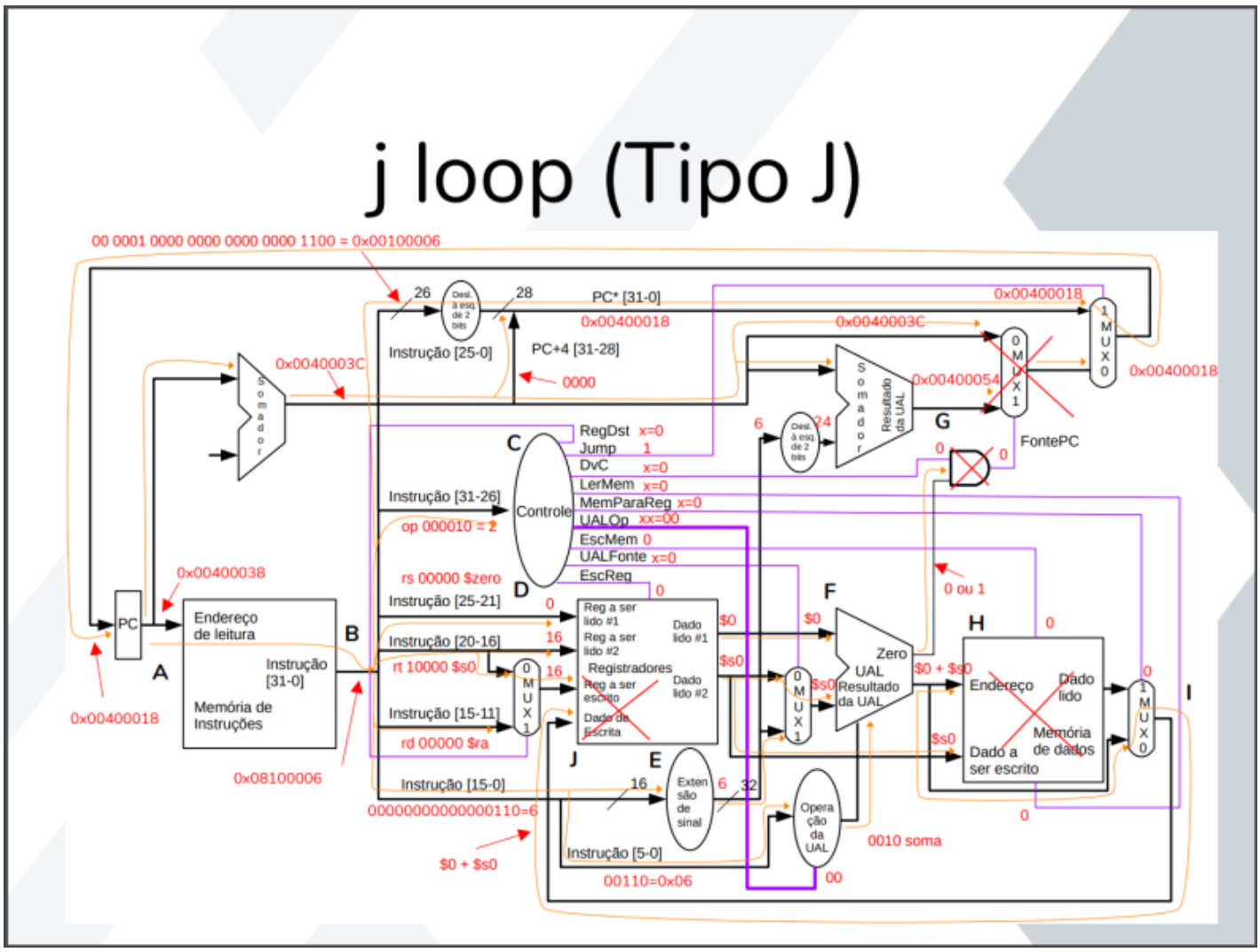
(H) A memória de dados recebe a subtração $s0-$t1 no barramento de endereços e o conteúdo do registrador $t1 na entrada “dado a ser escrito”. Esta memória recebe os sinais de controle EscMem = 0 e LerMem = 0.

(I) O multiplexador com o sinal MemParaReg = 0 tem na sua saída o conteúdo do endereço $s0-$t1, da memória de dados. Este sinal é levado até a entrada “dado de escrita” do banco de registradores.

(J) O sinal EscReg = 0 do banco de registradores não permite a escrita neste banco. Nesta borda de subida o registrador PC também é atualizado. Gravamos neste registrador o endereço da próxima instrução que será executada pelo processador, o endereço 0x0040 003C ou o endereço 0x0040 0014.

Questão 7

A seguir descrevemos o fluxo dos sinais no processador monociclo, na execução da instrução j loop:



(A) Na borda de subida do sinal de relógio, o registrador PC é carregado com o endereço da próxima instrução que será executada pelo processador monociclo. Neste problema, PC é carregado com o endereço 0x0040 0038. Este endereço vai até o barramento de endereços da memória de instruções e a entrada de um somador.

(B) Na saída da memória de instruções temos a instrução 0x08100006. Esta é a instrução j loop. Os vários campos desta instrução são separados: o campo imediato da instrução jump (instrução[25-0]), opcode (instrução[31-26]), registrador rs (instrução[25-21]), registrador rt (instrução[20-16]), registrador rd (instrução[15-11]) e campo imediato (instrução[15-0]).

(C) O campo opcode (instrução[31-26]) é decodificado pela unidade de controle. Sinais de controle são gerados. Estes sinais estão apresentados na tabela 1: RegDst = X = 0, Jump = 1, UALFonte = X = 0, MemParaReg = X = 0, EscReg = 0, LerMem = X = 0, EscMem = 0, DvC X = 0, UALOp XX = 00.

(D) No banco de registradores chegam os endereços dos registradores rs (instrução[25- 21]) = $zero e rt (instrução[20-16]) = $s0, na entrada dos endereços dos registradores que serão lidos. O conteúdo destes registradores aparecem nas saídas do banco de registradores “dado lido #1” e “dado lido #2”. O banco de registradores recebe o sinal de controle EscReg = 0. Neste problema, o endereço do registrador é a entrada 0 do MUX. Nesta entrada temos o registrador $s0. O sinal de controle RegDst = X = 0 seleciona esta entrada deste MUX.

(E) Ocorre a extensão do sinal no campo imediato (instrução[15-0]) = 6. Este sinal passa de 16 para 32 bits. O campo funct(instrução[5-0]) = 0x06 é extraído do campo imediato e ligado a entrada do circuito que gera o código de operação da UAL. Este circuito também recebe o sinal de controle UALOp = 00. Usando a tabela 2, vemos que é gerado o sinal de controle para a UAL 0010, solicitando uma adição para a ULA.

(F) Em uma das entradas da ULA temos um MUX que recebe o sinal de controle UALFonte = X = 0. É selecionado o “dado lido 2”, que é o registrador rt = $s0 para uma das entradas da ULA. Esta recebe em sua outra entrada o conteúdo do registrador rs = $zero. O sinal de controle 0010 do controle de operação da ULA faz com que as entradas sejam somadas. Na saída da ULA temos o valor $zero+$s0.

(G) Um segundo somador calcula o endereço de desvio de instruções condicionais beq. O valor imediato = 6 é deslocado por 2 bits para a esquerda, o que equivale a uma multiplicação por 4. O valor imediato multiplicado por 4, 18, é uma das entradas do somador. A outra entrada é o endereço da próxima instrução que será executada, PC+4 = 0x0040 003C. Na saída do somador temos o endereço 0x0040 0054. Na saída do somador temos um multiplexador com um sinal de controle FontePC. Uma porta AND gera o sinal FontePC, que é igual a 0, porque uma de suas entradas tem o sinal de controle DvC igual a 0. Neste caso, a saída da porta, o sinal FontePC será sempre 0, independente do valor do sinal zero vindo da ULA. Com FontePC = 0 selecionamos o endereço PC+4 = 0x0040 003C. Este endereço é a entrada 0 do MUX com o sinal de controle jump. Como este sinal de controle é 1, a saída deste multiplexador será o valor do endereço de desvio incondicional. O endereço de desvio incondicional vai para a entrada do registrador PC.

(H) A memória de dados recebe a soma $zero+$s0 no barramento de endereços e o conteúdo do registrador $s0 na entrada “dado a ser escrito”. Esta memória recebe os sinais de controle EscMem = 0 e LerMem = 0.

(I) O multiplexador com o sinal MemParaReg = 0 tem na sua saída o conteúdo do endereço $zero+$s0, da memória de dados. Este sinal é levado até a entrada “dado de escrita” do banco de registradores.

(J) O sinal EscReg = 0 do banco de registradores não permite a escrita neste banco. Nesta borda de subida o registrador PC também é atualizado. Gravamos neste registrador o endereço da próxima instrução que será executada pelo processador, o endereço 0x0040 0018.

Participantes: Ana Lilian Alfonso Toledo e Tobias Viero de Oliveira