

Questão 1: Escreva um procedimento, em assembly para o MIPS, para dividir dois números inteiros de 32 bits. Use o segundo algoritmo da divisão, apresentado em sala de aula. Escreva um programa, em assembly para o MIPS, usando este procedimento para realizar a divisão $x \div y$, com $x = 0x90357274$ e $y = 0x12341234$. Repita a divisão com $x = 0x12341234$ e $y = 0x90357274$. Mostre a saída da execução do seu programa no programa MARS. Verifique se o resultado da divisão apresentado pelo programa está correto.

O programa realiza o cálculo do resto e do quociente da divisão entre as variáveis globais x e y . Primeiramente, carregam-se os dois valores sem sinal para o registradores e chama-se o procedimento para calcular o quociente e o resto de sua divisão. O algoritmo faz uso de um procedimento auxiliar para realizar o deslocamento de 1 bit para a esquerda entre 2 registradores de 32 bits. Tal procedimento propaga o MSB do registrador considerado menos significativo para o LSB do registrador considerado mais significativo ao realizar o deslocamento. Após as iterações da divisão, o programa encerra com a impressão dos resultados.

O resultado da divisão entre $x = 0x90357274$ e $y = 0x12341234$ pode ser visualizado na figura 1. Também pode-se notar que tais valores estão fidedignos aos apresentados por uma calculadora na figura 2. Além disso, os resultados podem ser evidenciados na divisão entre $x = 0x12341234$ e $y = 0x90357274$ nas figuras 3 e 4.

```
x / y = 7
x % y = 281604872

-- program is finished running --
```

Figura 1: A saída do programa para a divisão entre $x = 0x90357274$ e $y = 0x12341234$

Input interpretation	Input interpretation
Quotient[90357274 ₁₆ , 12341234 ₁₆]	90357274 ₁₆ mod 12341234 ₁₆
Result	Result
7 ₁₆	10c8f308 ₁₆
Decimal form	Decimal form
7	281604872

(a) Quociente da divisão.

(b) Resto da divisão.

Figura 2: Resultados para a divisão entre $x = 0x90357274$ e $y = 0x12341234$ no WolframAlpha.

```
x / y = 0
x % y = 305402420

-- program is finished running --
```

Figura 3: A saída do programa para a divisão entre $x = 0x12341234$ e $y = 0x90357274$.

Input interpretation
Quotient[12341234 ₁₆ , 90357274 ₁₆]
Result
0 ₁₆
Decimal form
0

(a) Quociente da divisão.

Input interpretation
12341234 ₁₆ mod 90357274 ₁₆
Result
12341234 ₁₆
Decimal form
305 402 420

(b) Resto da divisão.

Figura 4: Resultados para a divisão entre $x = 0x12341234$ e $y = 0x90357274$ no WolframAlpha.

Questão 2: Escreva um procedimento double cos(double x), em assembly para o MIPS, para calcular o cosseno de um ângulo x, dado em radianos. O procedimento calcula o cosseno usando uma série de Taylor expandida em $x = 0$ (veja a equação 1). No procedimento, trunque a série em $n = 7$ (até o termo $\frac{x^{14}}{14!}$).

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2 \cdot n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \frac{x^{12}}{12!} - \frac{x^{14}}{14!} + \dots$$

Crie um programa em assembly para o MIPS. O programa permite a entrada de um ângulo x em graus, converte o ângulo para radianos, calcula o cosseno do ângulo usando o procedimento chama o procedimento $\cos(x)$ e imprime o resultado. Use o programa para calcular o seno de 40.67° . Mostre a saída da execução do seu programa no programa MARS. Verifique se o resultado apresentado pelo seu programa está correto.

Inicialmente, solicita-se ao usuário que insira um ângulo em graus, que é então convertido para radianos utilizando a devida fórmula de conversão. Em seguida, utiliza-se uma série de Taylor (até $n = 7$) para calcular o cosseno do ângulo, armazenando o resultado em registradores de ponto flutuante. Posteriormente, o seno é calculado com base na relação fundamental entre cosseno e seno. O programa encerra sua execução após a impressão dos resultados.

O resultado para o ângulo 40.67° pode ser visto na figura 5, também pode-se comparar o resultado do programa com a solução apresentada pelo WolframAlpha (figuras 6 e 7).

```
Digite o angulo em graus: 40.67
cos(x) = 0.7584756701844519
sen(x) = 0.6517013562501205

-- program is finished running --
```

Figura 5: Resultado apresentado pelo programa.

Input
cos(40.67 °)
Result
0.758476...

Figura 6: Resultado do cosseno.

Input
sin(40.67 °)
Result
0.651701...

Figura 7: Resultado do seno.

Questão 3: Represente o número $x = 114.55469$ em ponto flutuante, precisão simples. Mostre os passos na solução deste problema.

1. Converter o número para a sua representação binária.

- a) Conversão da parte fracionária.

$$0.55469 \cdot 2 = 1.10938$$

$$0.10938 \cdot 2 = 0.21876$$

$$0.21876 \cdot 2 = 0.43752$$

$$0.43752 \cdot 2 = 0.87504$$

$$0.87504 \cdot 2 = 1.75008$$

$$0.75008 \cdot 2 = 1.50016$$

$$0.50016 \cdot 2 = 1.00032$$

$$0.00032 \cdot 2 = 0.00064$$

$$0.00064 \cdot 2 = 0.00128$$

$$0.00128 \cdot 2 = 0.00256$$

$$0.00256 \cdot 2 = 0.00512$$

$$0.00512 \cdot 2 = 0.01024$$

$$0.01024 \cdot 2 = 0.02048$$

$$0.02048 \cdot 2 = 0.04096$$

$$0.04096 \cdot 2 = 0.08192$$

$$0.08192 \cdot 2 = 0.16384$$

$$0.16384 \cdot 2 = 0.32768$$

$$0.32768 \cdot 2 = 0.65536$$

$$0.65536 \cdot 2 = 1.31072$$

$$0.31072 \cdot 2 = 0.62144$$

$$0.62144 \cdot 2 = 1.24288$$

$$0.24288 \cdot 2 = 0.48576$$

$$0.48576 \cdot 2 = 0.97152$$

$$0.97152 \cdot 2 = 1.94304$$

⋮

$$0.55469 = 0.\underbrace{1000111000000000000101001}_{(24\text{ bits para o significando})}$$

b) Conversão da parte inteira.

$$\begin{array}{r}
 114 \\
 \underline{-114} \quad 2 \\
 \hline
 0 \quad 57 \quad 2 \\
 \underline{-56} \quad 28 \quad 2 \\
 \hline
 1 \quad 28 \quad 14 \quad 2 \\
 \underline{-28} \quad 0 \quad \underline{-14} \quad 7 \\
 \hline
 0 \quad 14 \quad 0 \quad 2 \\
 \underline{-14} \quad \underline{-7} \quad \underline{-2} \quad 1 \\
 \hline
 0 \quad 0 \quad 0 \quad 1
 \end{array}$$

$$114 = 1110010_2$$

2. Ajustar o número para o significando.

$$+1110010.100011100000000000101001_2 \cdot 2^0$$

$+1.$ 11001010001110000000000 $0101001_2 \cdot 2^{6=E \rightarrow \text{expoente}}$

$F \rightarrow \text{significando}$

3. Calcular o expoente polarizado.

$$EP = E + P$$

$EP = 6 + 127$ (o peso para precisão simples é 127, Padrão IEEE 754-2008)

$$EP = 133$$

$$\begin{array}{r}
 133 \\
 \underline{-132} \\
 1 \\
 \end{array}
 \quad
 \begin{array}{r}
 2 \\
 \underline{66} \\
 66 \\
 \underline{33} \\
 32 \\
 \end{array}
 \quad
 \begin{array}{r}
 2 \\
 \underline{16} \\
 16 \\
 \underline{8} \\
 0 \\
 \end{array}
 \quad
 \begin{array}{r}
 2 \\
 \underline{4} \\
 4 \\
 \underline{0} \\
 0 \\
 \end{array}
 \quad
 \begin{array}{r}
 2 \\
 \underline{2} \\
 2 \\
 \underline{0} \\
 1 \\
 \end{array}$$

$$133 = 10000101_2$$

4. Representação em ponto flutuante de precisão simples.

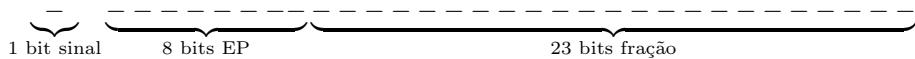
$$\underbrace{0}_{+} \underbrace{10000101}_{EP} \underbrace{11001010001110000000000_2}_{\text{Fração}} = 0x42E51C00$$

Questão 4: Qual o valor decimal do número $x = 0x34343400$, representado em ponto flutuante, precisão simples. Mostre os passos na solução deste problema.

- O número $0x34343400$ em hexadecimal é representado na base binária por:

$$00110100001101000011010000000000_2$$

- Como estamos analisando um ponto flutuante de 32 bits, ou seja, de precisão simples, o número seguirá o seguinte formato.



- Portanto, posicionamos nosso número nesse formato:

$$\underbrace{0}_{S=0} \underbrace{01101000}_{EP=104} \underbrace{0110100001101000000000}_{0.F=0.4078369140625}$$

- Convertemos o número para decimal com a fórmula:

$$\begin{aligned}
 & (-1)^s \cdot (1 + 0.F) \cdot 2^{EP-P} = \\
 & (-1)^0 \cdot (1 + 0.4078369140625) \cdot 2^{104-127} = \\
 & 1 \cdot (1 + 0.4078369140625) \cdot 2^{-23} = \\
 & 1.4078369140625 \cdot 2^{-23} = \\
 & 1.67827238328754901885986328125 \cdot 10^{-7} = \\
 & 0.000000167827238328754901885986328125
 \end{aligned}$$

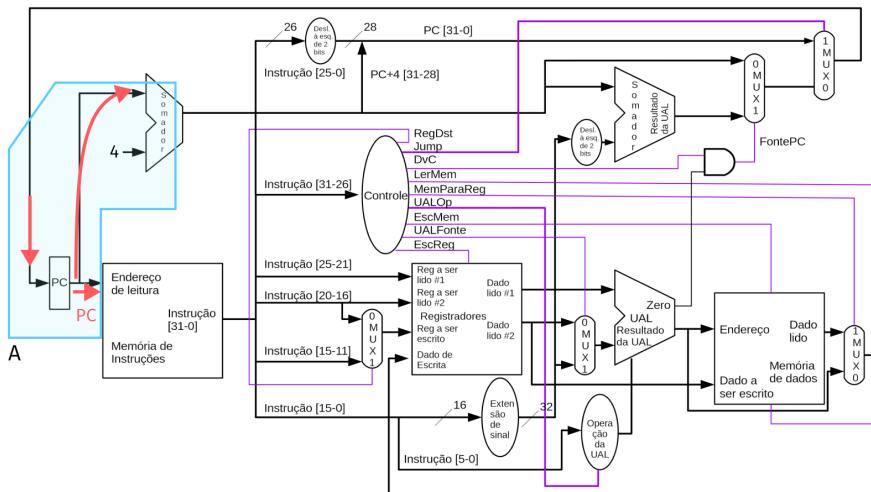
Questão 5: Explique detalhadamente, usando as tabelas 1 e 2 e o diagrama de blocos do processador na figura 1, como a instrução sw \$a0, 48(\$at) é executada pelo processador monociclo. Converta a instrução para linguagem de máquina, apresentando os campos. Apresente na figura os sinais de controle. Escreva um texto explicando como a instrução é executada.

sw \$rt, imm(\$rs)

sw \$a0, 48(\$at) = 101011 00001 00100 00000000000110000
opcode rt rs imm

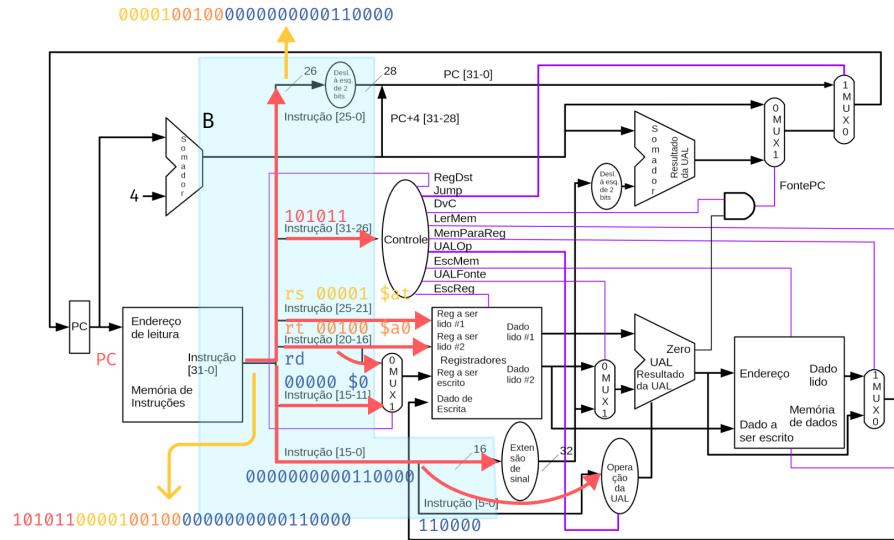
A seguir descrevemos o fluxo dos sinais no processador monociclo, na execução da instrução sw \$a0, 48(\$at):

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



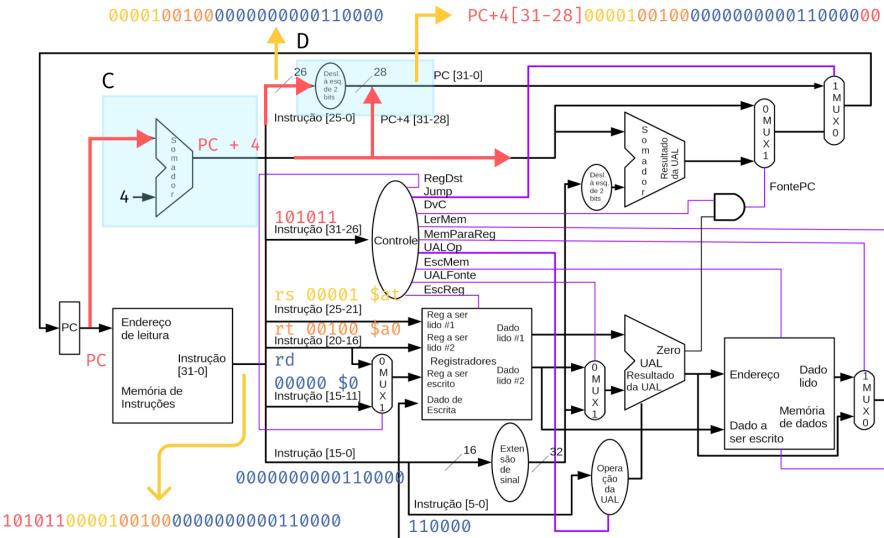
B) Na saída de memória, encontra-se a instrução sw \$a0, 48(\$at) em seu formato binário. Os campos da instrução são separados:

- campo OPCODE (Instrução [31 - 26]) = 101011
- registrador RS (Instrução [25 - 21]) = 00001
- registrador RT (Instrução [20 - 16]) = 00100
- registrador RD (Instrução [15 - 11]) = 00000
- campo IMEDIATO 16 (Instrução [15 - 0]) = 000000000000110000
- campo FUNCT (Instrução [5 - 0]) = 110000
- campo JUMP (Instrução [25 - 0]) = 000010010000000000000000110000



C) Adiciona-se 4 ao valor de PC.

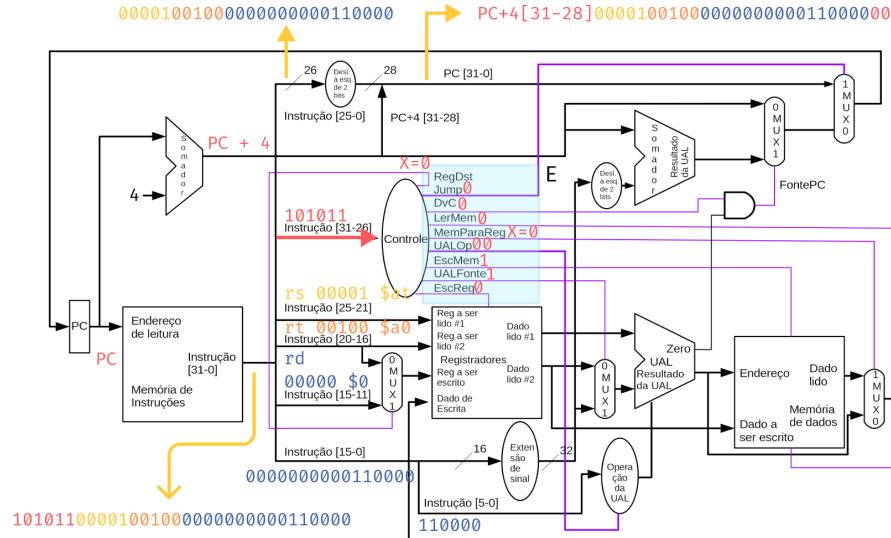
D) Os 26 bits menos significativos da instrução são deslocados 2 bits para a esquerda. Após isso, os 4 bits mais significativos de $PC + 4$ são concatenados com esses 28 bits. Forma-se, assim, o endereço de desvio incondicional.



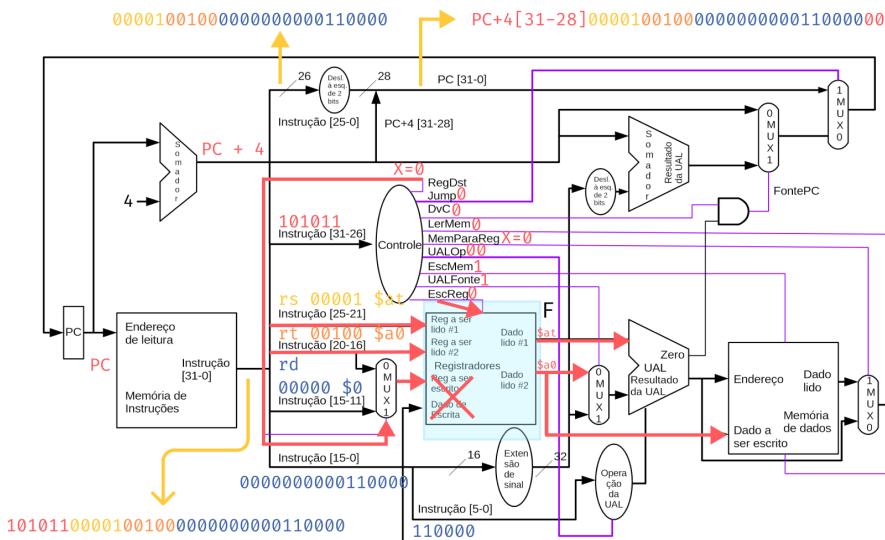
E) O campo OPCODE é decodificado pela unidade de controle e gera-se os seguintes sinais para o processador:

- $\text{RegDst} = 0$
 - $\text{Jump} = 0$
 - $\text{DvC} = 0$
 - $\text{LerMem} = 0$

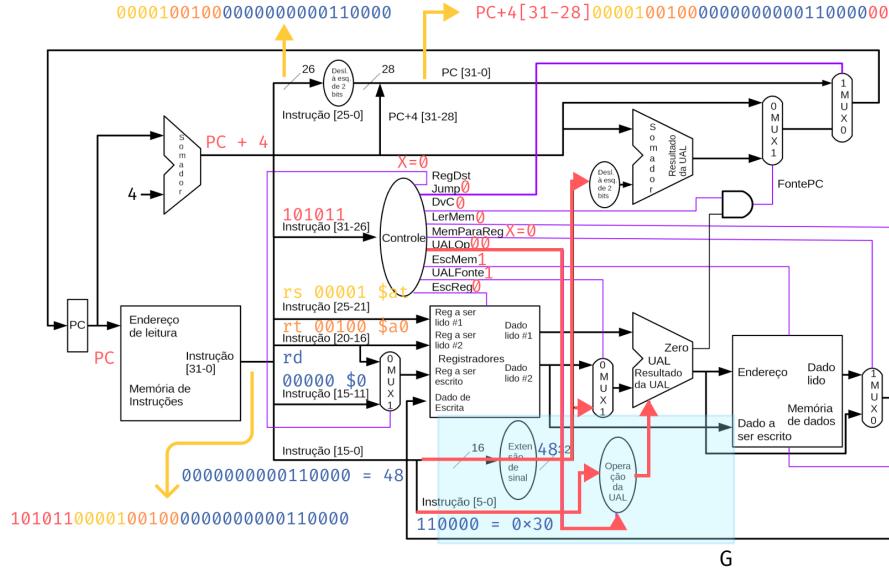
- MemParaReg = 0
- UALOp = 0
- EscMem = 1
- UALFonte = 1
- EscReg = 0



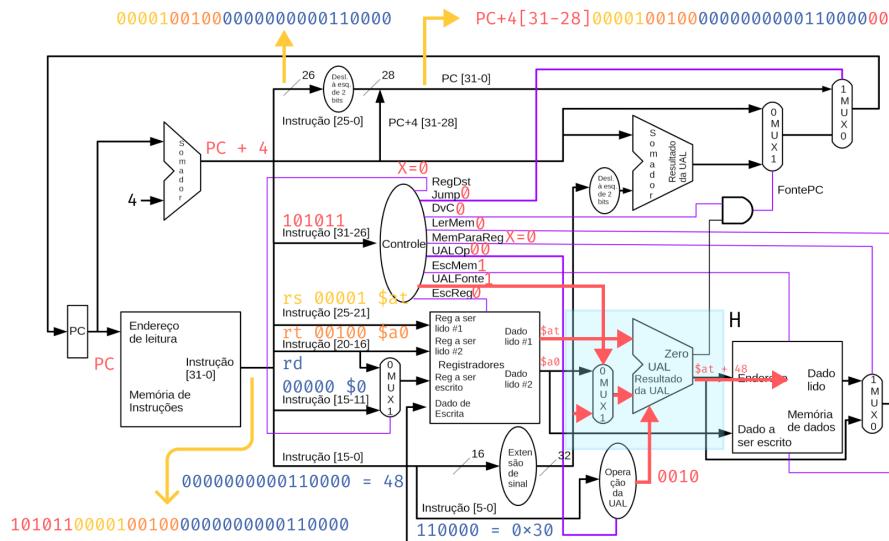
F) No banco de registradores, chegam os endereços de rs = \$at ao "reg a ser lido #1" e de rt = \$a0 ao "reg a ser lido #2". Os conteúdos desses registradores aparecem nas saídas do banco de registradores em "dado lido #1" e em "dado lido #2", respectivamente. O "reg a ser escrito" e o "dado de escrita" são ignorados, pois o sinal de controle EscReg é igual a 0. A saída multiplexador conectado à "reg a ser escrito" é irrelevante por não ser realizada nenhuma escrita aos registradores.



G) 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.

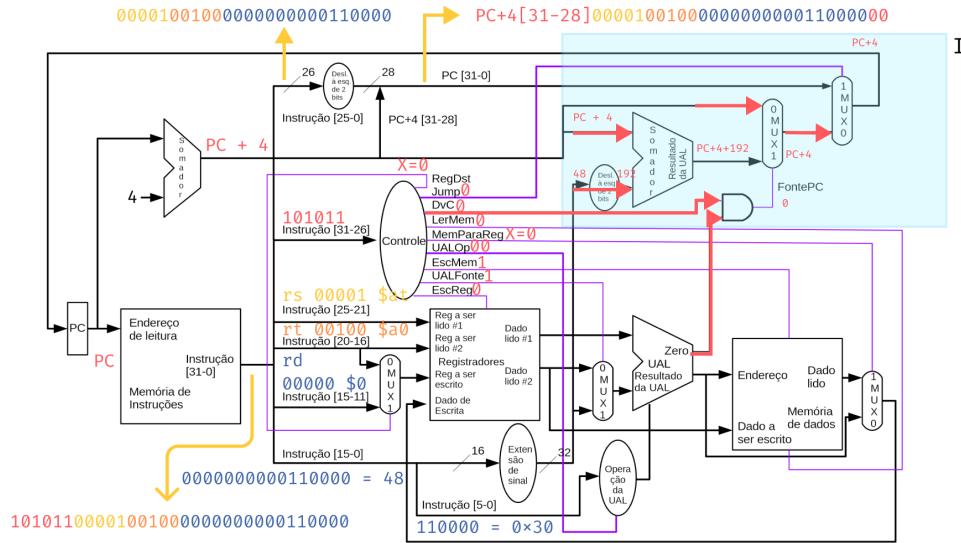


H) Em uma das entradas da ULA temos um MUX que recebe o sinal de controle UALFonte = 1. Desse modo, é selecionado o valor imediato estendido 48 para uma das entradas da ULA. Esta recebe em sua outra entrada o conteúdo do registrador rs = \$at. O sinal de controle 0010 do controle de operação da ULA faz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

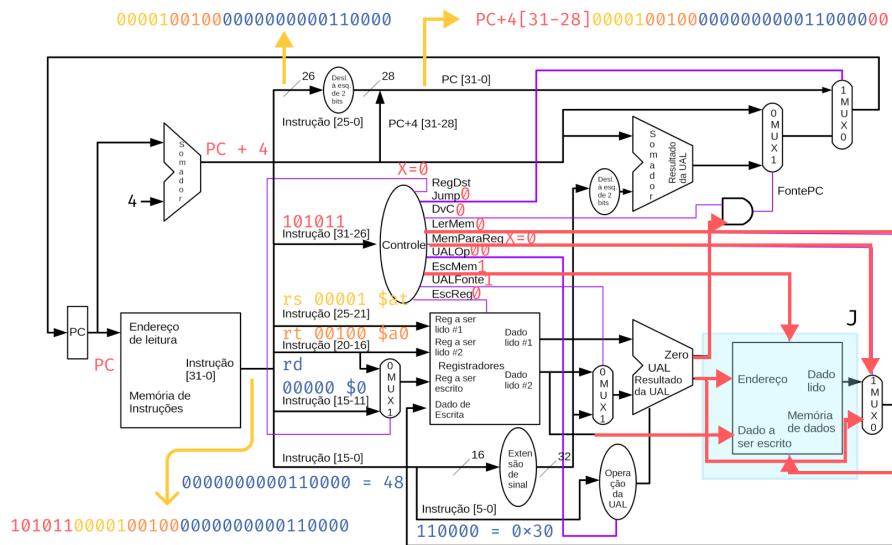


I) Um segundo somador calcula o endereço de desvio de instruções condicionais beq. O valor imediato = 48 é deslocado por 2 bits para a esquerda, o que equivale a uma multiplicação por 4. O valor imediato multiplicado por 4, 192, é uma das entradas do somador. A outra entrada é o endereço da

próxima instrução que será executada, PC + 4. Na saída do somador, temos o endereço de PC + 4 + 192, conectado a um multiplexador com o 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. 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.

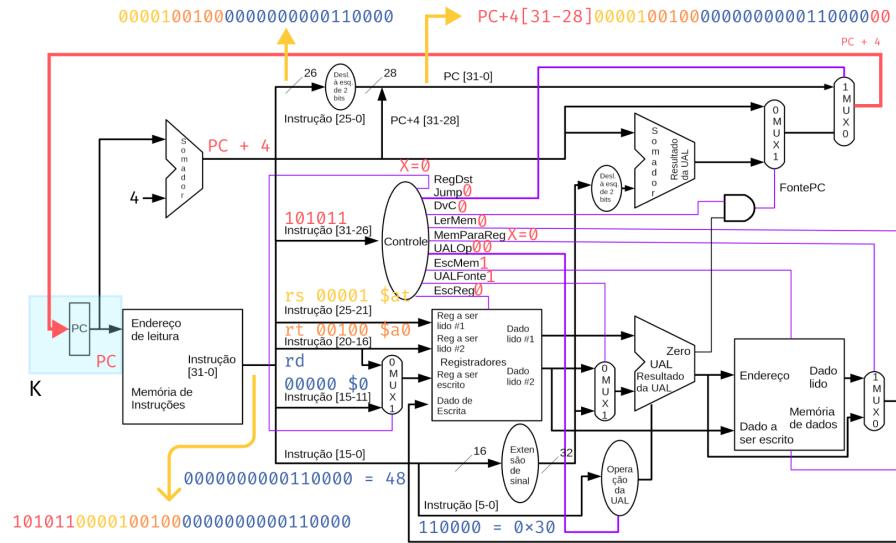


J) A memória de dados recebe a soma $\$at + 48$ no barramento de endereços e o conteúdo do registrador $\$a0$ na entrada “dado a ser escrito”. Esta memória recebe os sinais de controle EscMem = 1 e LerMem = 0. Ocorre a escrita do valor na memória. A saída do o multiplexador que recebe MemParaReg é irrelevante, visto que nenhum valor será escrito aos registradores.



K) Na próxima borda de subida, o registrador PC será atualizado. Gravaremos neste registrador

o endereço da próxima instrução que será executada pelo processador.



Questão 6: Explique detalhadamente, usando as tabelas 1 e 2 e o diagrama de blocos do processador na figura 1, como a instrução beq \$s0, \$t1, loop é executada pelo processador monociclo. O endereço desta instrução é 0x004000038 e loop é um rótulo para o endereço 0x00400014. Converta a instrução para linguagem de máquina, apresentando os campos. Apresente na figura os sinais de controle. Escreva um texto explicando como a instrução é executada.

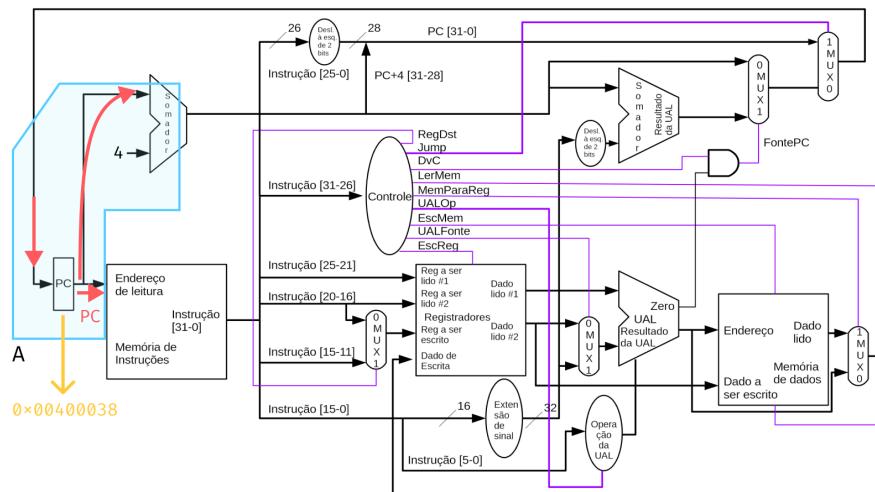
Um valor do endereço da instrução com 9 dígitos (0x004000038) é incomum de ser encontrado na arquitetura MIPS de 32 bits. Por parecer um erro de digitação e também por não ser possível realizar um desvio condicionado de $(0x004000038 + 4)$ para $0x00400014$, consideramos o endereço somente como 0x00400038.

beq \$rs, \$rt, imm

beq \$s0, \$t1, loop = 000100 10000 01001 111111111110110
 opcode rt rs imm

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

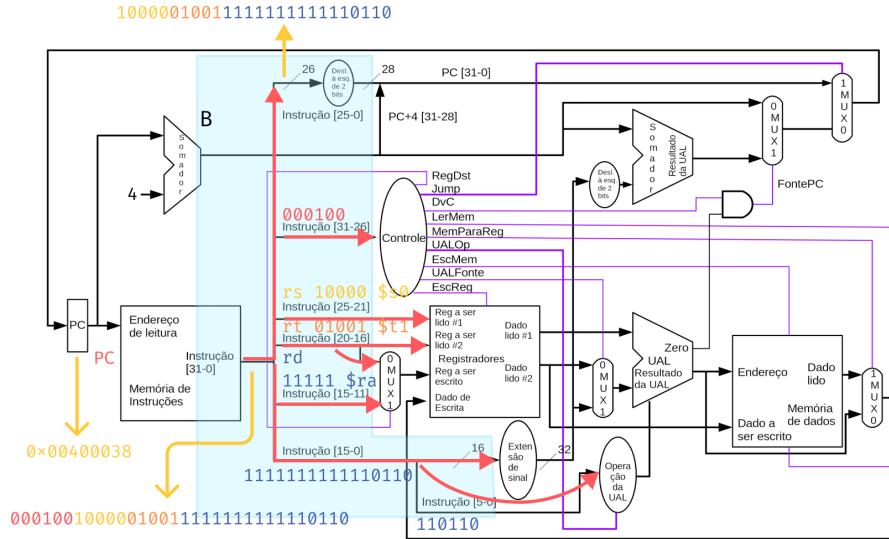
A) Na borda de subida do sinal de relógio, o registrador PC é carregado com o endereço da próxima instrução que será executada pelo processador monociclo. Neste problema, PC é carregado com o endereço 0x00400038. Este endereço vai até o barramento de endereços da memória de instruções e a entrada de um somador.



B) Na saída da memória, encontra-se a instrução beq \$s0, \$t1, loop em seu formato binário. Os campos da instrução são separados:

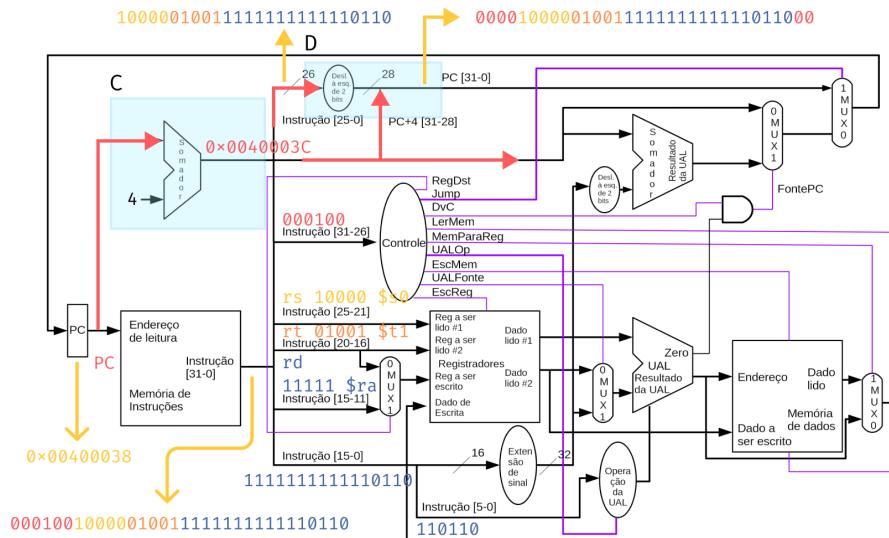
- campo OPCODE (instrução[31 - 26]) = 000100
- registrador RS (instrução[25 - 21]) = 10000
- registrador RT (instrução[20 - 16]) = 01001

- registrador RD (instrução[15 - 11]) = 11111
- campo IMEDIATO 16 (instrução[15 - 0]) = 1111111111110110
- campo FUNCT (instrução [5 - 0]) = 110110
- campo JUMP (instrução[25 - 0]) = 1000001001111111111110110



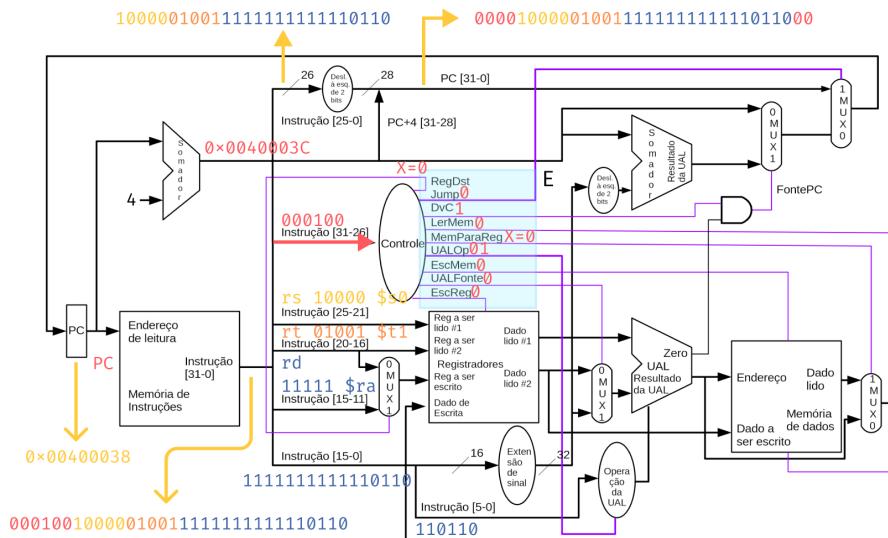
C) Adiciona-se 4 ao PC = 0x00400038 em um somador, resultando em 0x0040003C em sua saída.

D) Os 26 bits menos significativos da instrução (10 0000 1001 1111 1111 1111 0110) são deslocados 2 bits para a esquerda (1000 0010 0111 1111 1111 1101 1000). Após isso, os 4 bits mais significativos de PC + 4 (0000) são concatenados aos 28 bits (0000 1000 0010 0111 1111 1111 1101 1000). Forma-se, assim, o endereço de desvio incondicional.

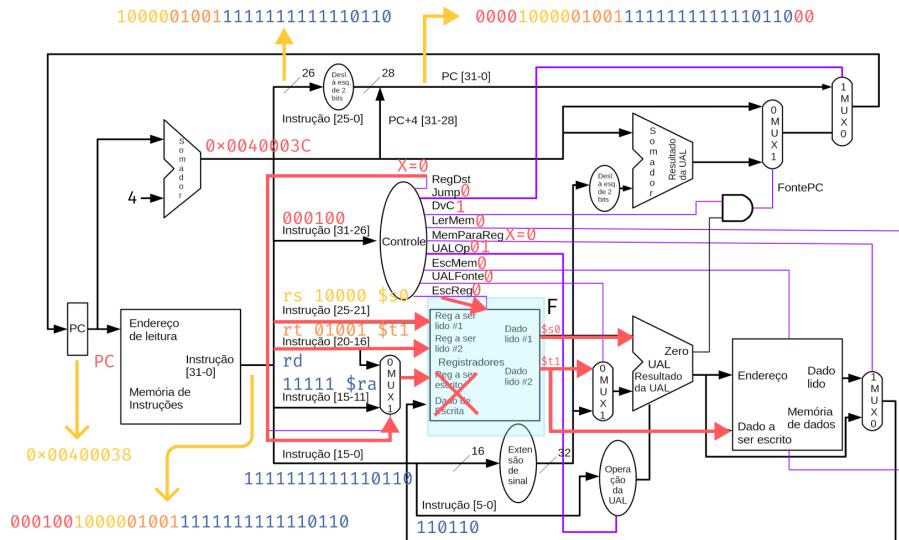


E) O campo OPCODE é decodificado pela unidade de controle e gera-se os seguintes sinais para o processador:

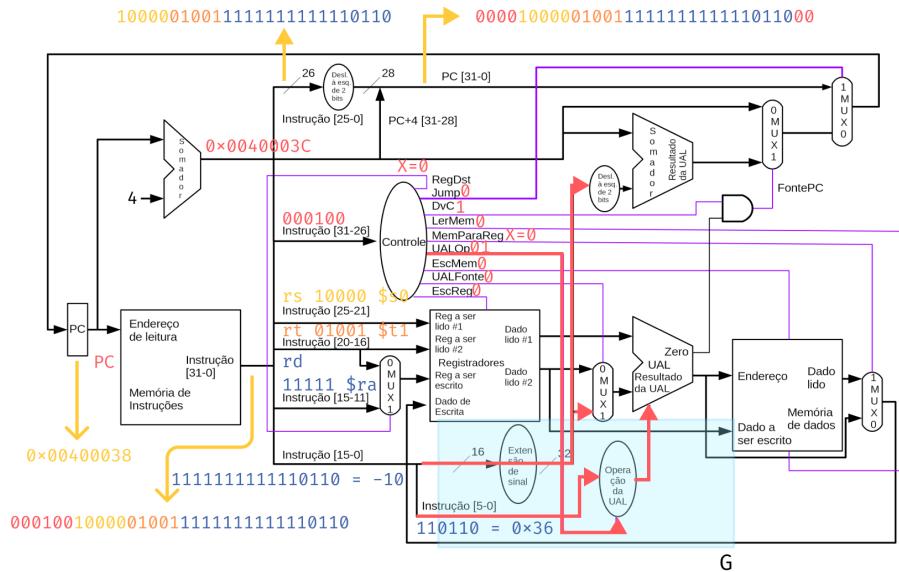
- RegDst = 0
 - Jump = 0
 - DvC = 1
 - LerMem = 0
 - MemParaReg = 0
 - UALOp = 01
 - EscMem = 0
 - UalFonte = 0
 - EscReg = 0



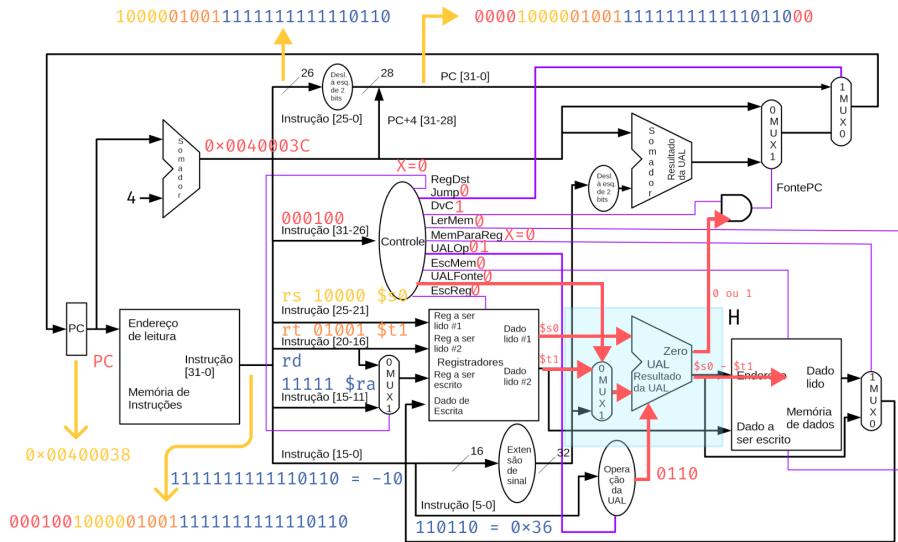
F) No banco de registradores, chegam os registradores $rs = \$s0$ ao "reg a ser lido #1" e $rt = \$t1$ ao "reg a ser lido #2". Os conteúdos desses registradores aparecem nas saídas do banco de registradores em "dado lido #1" e em "dado lido #2", respectivamente. O "reg a ser escrito" e o "dado de escrita" são ignorados pois $EscReg = 0$. Logo, a saída do multiplexador selecionado por $RegDst$ é irrelevante.



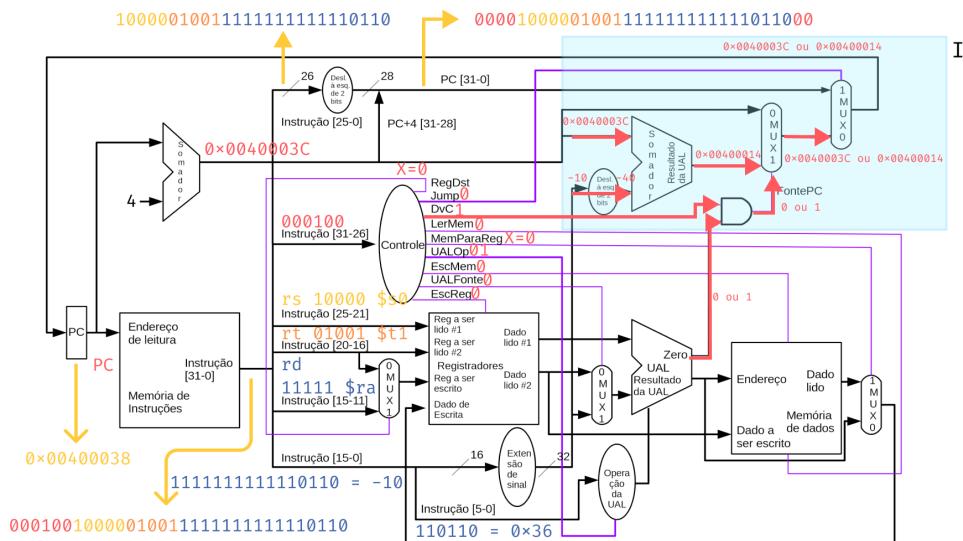
G) Ocorre a extensão de sinal no campo imediato (instrução[15 - 0]) = -10. Este sinal passa de 16 bits para 32 bits. O campo funct (instrução [5 - 0]) = 0x36 é extraído e ligado a entrada do circuito que gera o código da operação UAL. Este circuito também recebe o sinal de controle UALOp = 01, solicitando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



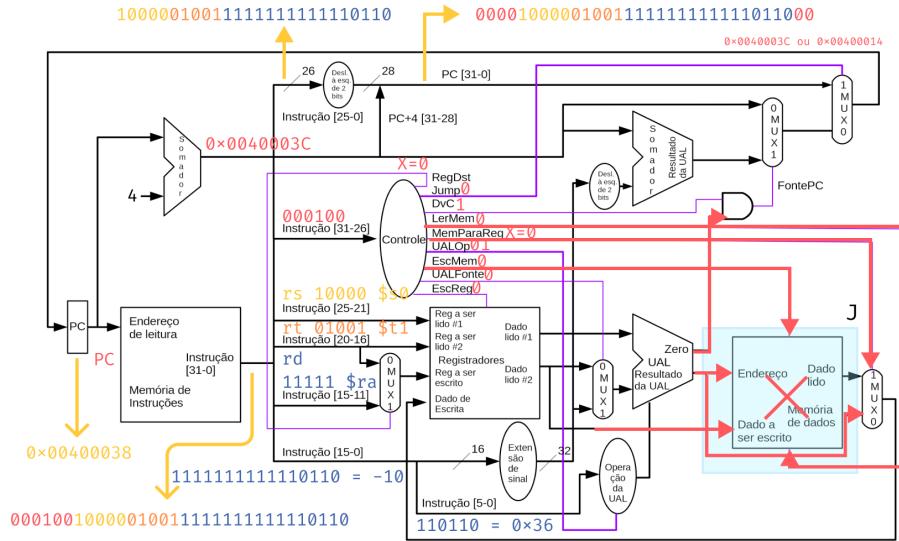
H) Em uma das entradas da ULA, temos um MUX que recebe o sinal de controle UALFonte = 0. Desse modo, é selecionado o conteúdo armazenando no registrador rt = \$t1. O sinal de controle 0110 da ULA faz com que as entradas sejam subtraídas. Na saída da ULA, temos que a flag Zero será 1 quando as entradas forem iguais e 0 quando as entradas forem diferentes. O resultado da ULA é irrelevante, pois EscMem = 0, LerMem = 0 e EscReg = 0.



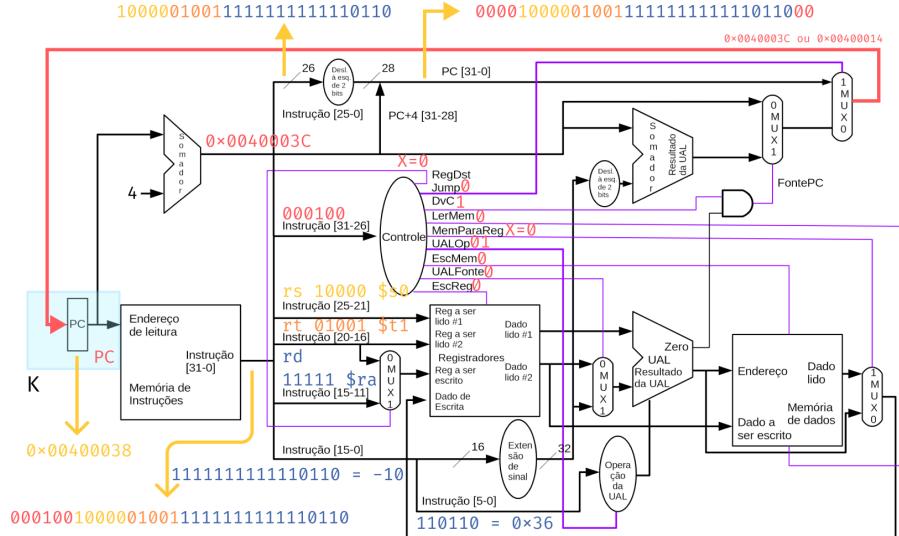
I) Um segundo somador calcula o endereço de desvio de instruções condicionadas, como beq. O valor imediato $= -10$ é deslocado por 2 bits para a esquerda, o que equivale a uma multiplicação por 4. O resultado dessa operação, -40 , é uma das entradas do somador. A outra entrada é a próxima instrução que será executada, $PC + 4 = 0x0040003C$. Na saída do somador, temos o endereço de $PC + 4 - 40 = 0x00400014$ conectado a um multiplexador com um sinal de controle FontePC. Uma porta AND gera o sinal FontePC que, em virtude de entrada DvC = 1, dependerá somente da flag Zero do somador, isto é, da igualdade dos conteúdos em $rs = \$s0$ e em $rt = \$t1$. Logo, o endereço da entrada 0 do MUX com o sinal de controle Jump estará condicionado a igualdade entre os conteúdos dos registradores rs e rt . Consequentemente, sabendo que $Jump = 0$, o endereço para a próxima instrução também dependerá da condição.



J) Como os sinais de controle $LerMem = 0$ e $EscMem = 0$, a memória de dados não realiza nenhuma operação e ignora a entrada de endereço e a entrada de dado a ser escrito. A saída do multiplexador controlado por $MemParaReg$ é irrelevante, visto que $EscReg = 0$ e nenhum valor será escrito aos registradores.



K) Na próxima borda de subida, o registrador PC será atualizado. Caso os registradores rs = \$s0 e rt = \$t1 forem iguais, gravaremos o endereço do desvio condicional (0x00400014), senão gravaremos o endereço instrução subsequente (0x0040003C). O endereço gravado será a próxima instrução executada pelo processador.



Questão 7: Explique detalhadamente, usando as tabelas 1 e 2 e o diagrama de blocos do processador na figura 1, como a instrução j loop é executada pelo processador monociclo. O endereço desta instrução é 0x004000038 e loop é um rótulo para o endereço 0x00400018. Converta a instrução para linguagem de máquina, apresentando os campos. Apresente na figura os sinais de controle. Escreva um texto explicando como a instrução é executada.

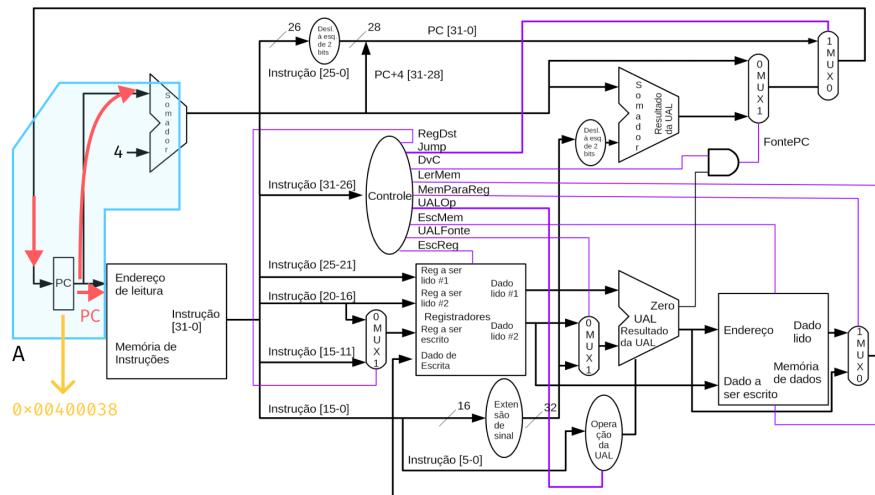
Como na questão anterior havia o mesmo endereço com 9 dígitos (0x004000038), também consideramos o endereço 0x00400038 para esta questão.

j label

$$j \text{ loop} = \underbrace{000010}_{\text{opcode}} \underbrace{00000100000000000000000000110}_{\text{endereço de jump}}$$

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

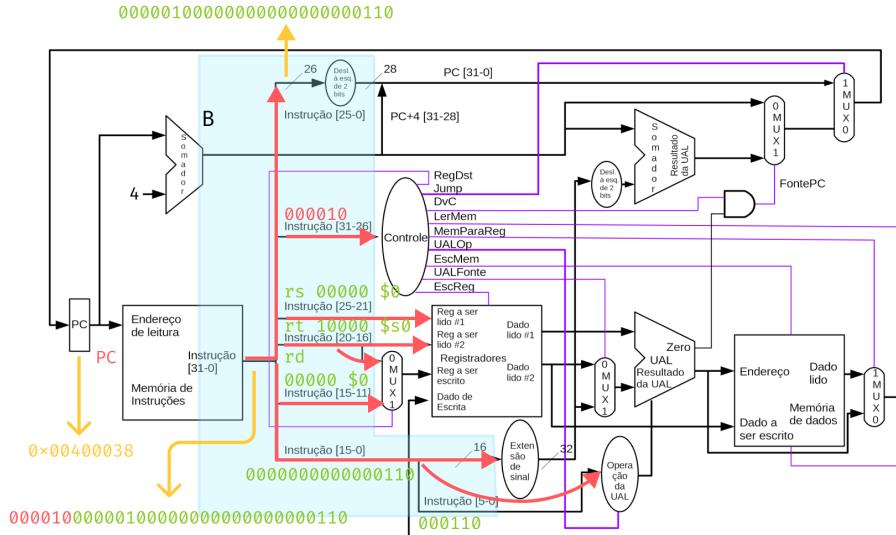
A) Na borda de subida do sinal de relógio, o registrador PC é carregado com o endereço da próxima instrução que será executada pelo processador monociclo. Neste problema, PC é carregado com o endereço 0x00400038. Este endereço vai até o barramento de endereços da memória de instruções e a entrada de um somador.



B) Na saída da memória, encontra-se a instrução j loop, em seu formato binário. Os campos da instrução são separados:

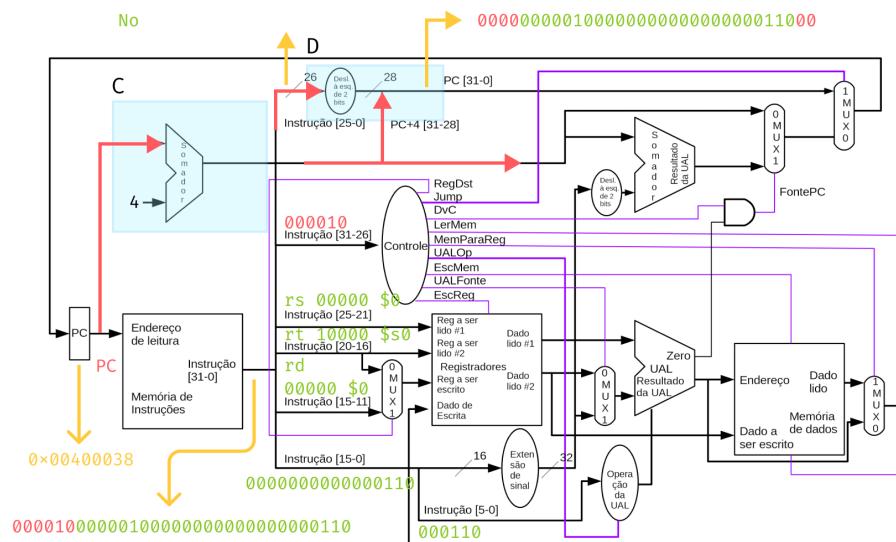
- campo OP CODE (instrução [31-26]) = 000010
- campo RS (instrução [25-21]) = 00000
- campo RT (instrução [20-16]) = 10000
- campo RD (instrução [15-11]) = 00000

- campo IMEDIATO (instrução [15-0]) = 0000000000000000110
- campo FUNCT (instrução [5-0]) = 000110
- campo JUMP (instrução [25-0]) = 00000100000000000000000000000000110



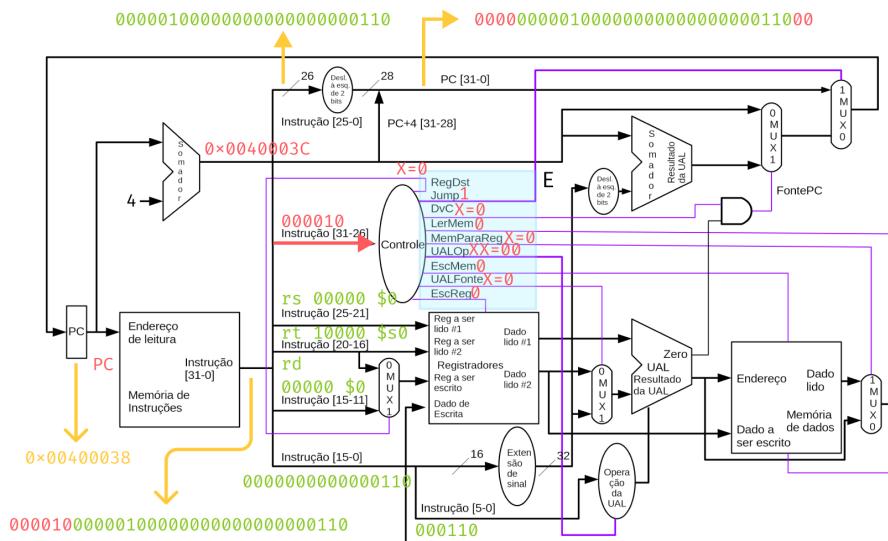
C) Adiciona-se 4 ao PC = 0x00400038 em um somador, resultando em 0x0040003C em sua saída.

D) Os 26 bits menos significativos da instrução (00 0001 0000 0000 0000 0000 0110) são deslocados 2 bits para a esquerda (0000 0100 0000 0000 0000 0001 1000). Após isso, os 4 bits mais significativos de PC+4 (0000) são concatenados aos 28 bits (0000 0100 0000 0000 0000 0001 1000). Forma-se assim, o endereço de desvio incondicional.

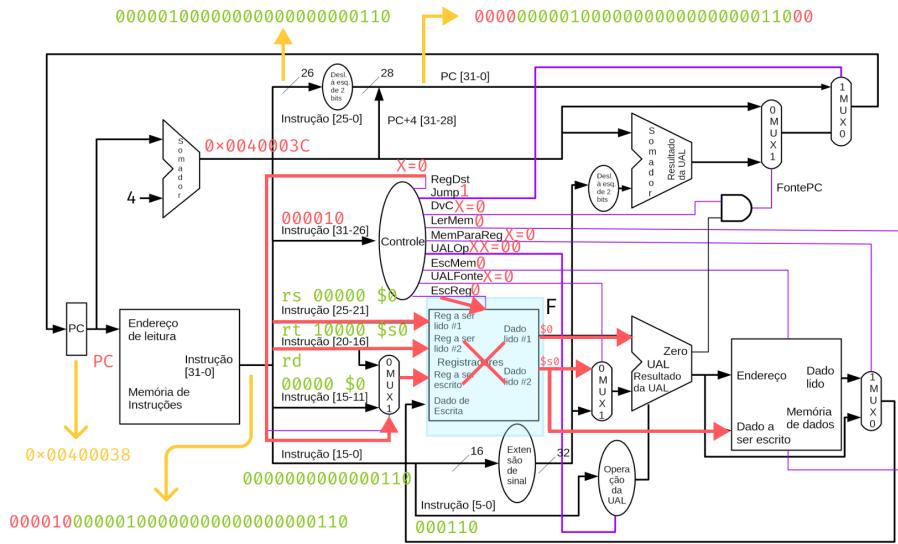


E) O campo OPCODE é decodificado pela unidade de controle e gera-se os seguintes sinais para o processador:

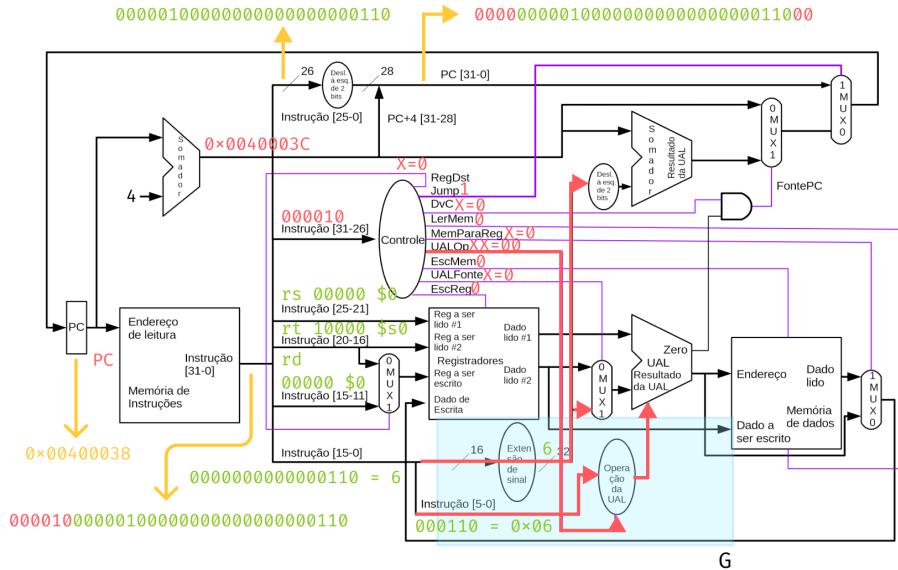
- RegDst = 0
- Jump = 1
- DvC = 0
- LerMem = 0
- MemParaReg = 0
- UALOp = 00
- EscMem = 0
- UalFonte = 0
- EscReg = 0



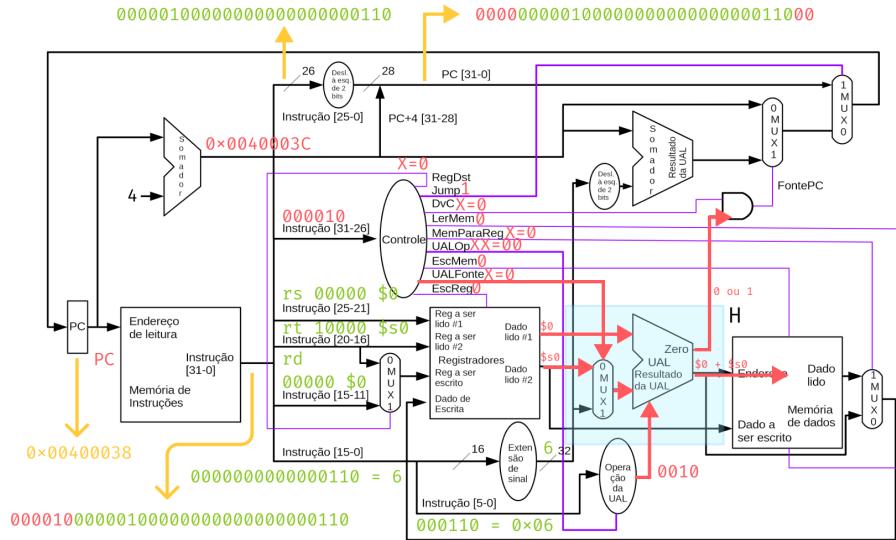
F) No banco de registradores, chegam os registradores $rs = \$0$ ao "reg a ser lido #1" e $rt = \$s0$ ao "reg a ser lido #2". Os conteúdos desses registradores aparecem nas saídas do banco de registradores em "dado lido #1" e em "dado lido #2", respectivamente, todavia, para a instrução atual são irrelevantes. O "reg a ser escrito" e o "dado de escrita" são ignorados pois $EscReg = 0$. Logo, a saída do multiplexador selecionado por $RegDst$ é irrelevante.



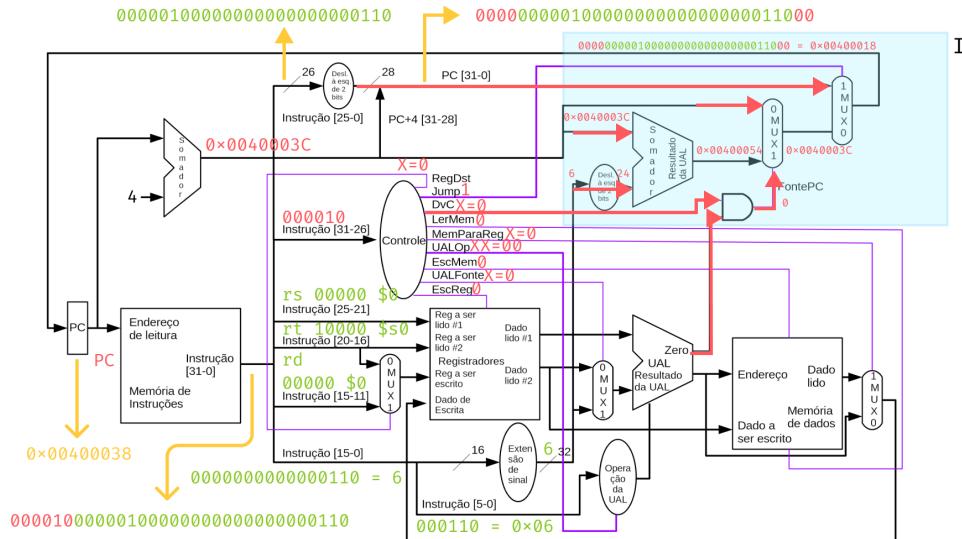
G) Ocorre a extensão de sinal no campo imediato (instrução [15-0]) = 6. Esse sinal passa de 16 para 32 bits. O campo FUNCT (instrução[5-0]) é extraído e ligado a entrada do circuito que gera o código da operação UAL. Este circuito também recebe o sinal de controle UALOp = 00, requisitando uma soma, no entanto esse sinal e a operação de soma são insignificantes para a instrução.



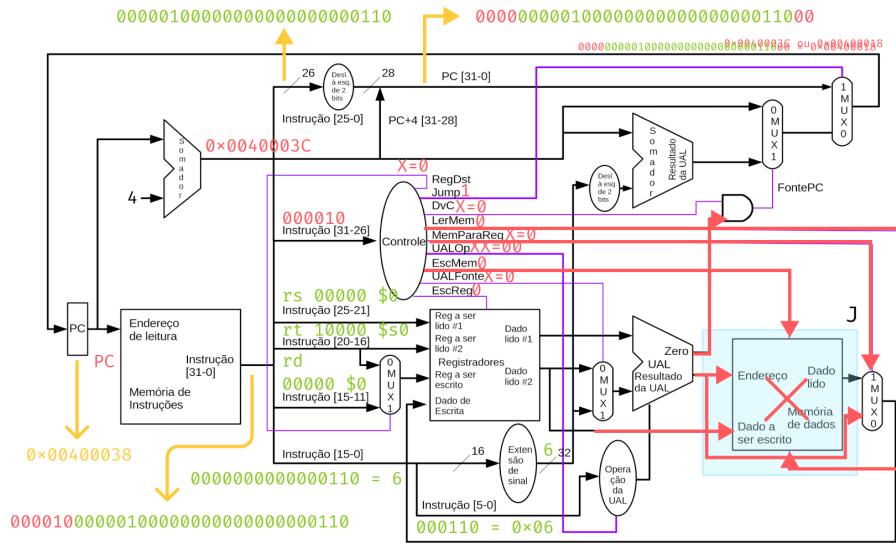
H) Em uma das entradas da ULA, temos um MUX que recebe o sinal de controle UALFonte = 0. Seleciona-se o conteúdo armazenado em rt = \$s0. O sinal de controle 0010 faz com que as entradas sejam somadas. O resultado da ULA é irrelevante, pois EscMem = 0, LerMem = 0 e EscReg = 0.



I) Um segundo somador calcula o endereço de desvio de instruções condicionadas, como o beq. O valor imediato = 6 é deslocado 2 bits para esquerda, o que equivale a uma multiplicação por 4. O resultado dessa operação, 24, é uma das entradas do somador. A outra entrada é a próxima instrução que será executada, $PC + 4 = 0x0040003C$. Na saída do somador temos o endereço de $PC + 4 + 24 = 0x00400054$ conectado a um multiplexador com um sinal de controle FontePC. Uma porta AND gera o sinal de FontePC. Esse valor é irrelevante, visto que o sinal de controle Jump = 1, selecionando a saída 1 do próximo multiplexador, que é o endereço $0x00400018$.



J) Como os sinais de controle $LerMem = 0$ e $EscMem = 0$, a memória de dados não realiza nenhuma operação e ignora a entrada de endereço e o dado a ser escrito. A saída do multiplexador controlado por $MemParaReg$ é irrelevante, visto que $EscReg = 0$ e nenhum valor será escrito nos registradores.



K) Na próxima borda de subida, o registrador PC será atualizado, gravando o endereço do desvio incondicional 0x00400018. Tal endereço será a próxima instrução executada pelo processador.

