Documentação Projeto 2 de C208L1

Nome: Joao Pedro Maciel de Souza

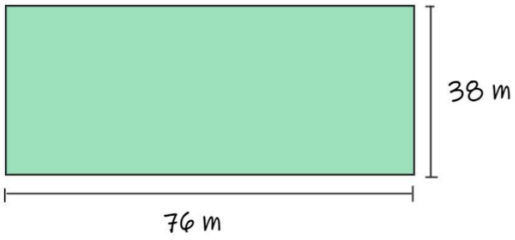
Matricula: 147

Curso: GES

Monitor: Thiago da Rocha Miguel

Problema a ser resolvido:

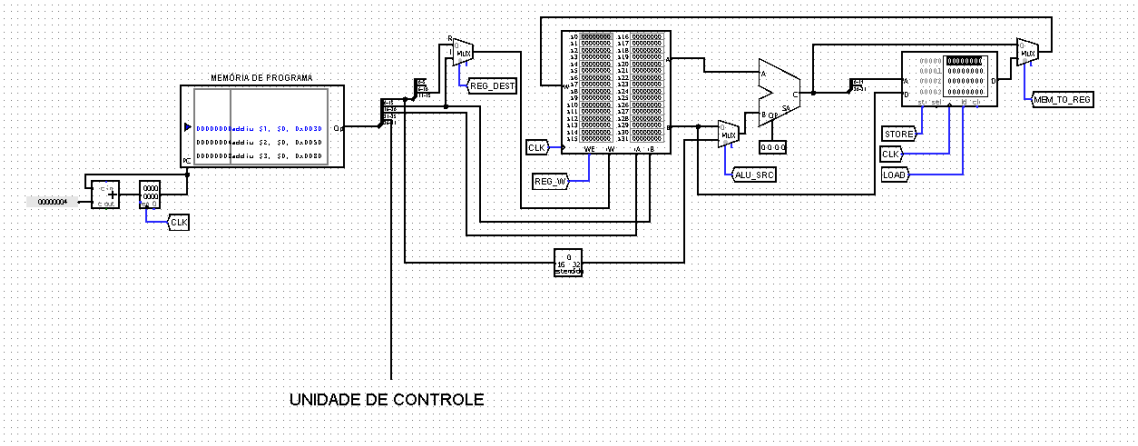
2. Cálculo de Perimetro com Operação Lógica:



Calcule a perimetro do retângulo apresentado. Com o resultado, aplique a lógica "E" utilizando o número 255 e instruções do tipo I.

Objetivo: criar um circuito logico para fazer o papel da unidade de controle, só que especificamente para resolver o problema proposto. Sabe-se que a Unidade de Controle é responsável por enviar os sinais aos bits de controle para que cada instrução possa ser corretamente executada, nesse caso ela precisa ser capaz apenas de lidar com quaisquer instruções utilizadas e não todas possíveis.

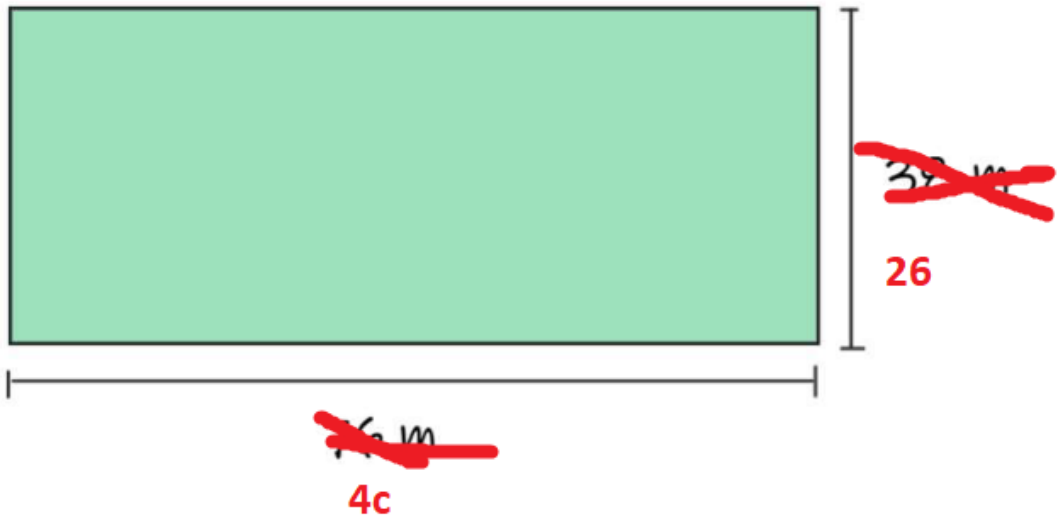
Circuito utilizado do processador como base:



Circuito desenvolvido durante as aulas teóricas de C208 juntamente com o professor Yvo, arquitetura MIPS sem JUMP e BRANCH.

Analisando o problema: para calcular o perímetro de um retângulo com tamanho dos lados definidos a priori precisamos apenas de somas imediatas. Tendo o resultado final armazenado num registrador temporário, basta usar uma operação logica do tipo "AND" nesse valor e armazenar esse resultado.

Convertendo os lados para hexadecimal temos:



Usaremos esses valores no momento do código, já que fazemos as operações em hexadecimal.

Circuito logico: para montar o circuito da unidade de controle precisamos considerar apenas as instruções ADDIU e ANDI. Iremos consultar seus OPCODEs e, com a ajuda de um distribuidor, alimentar as entradas de uma porta AND de 6 bits customizada (negando as entradas onde o bit do OPCODE é 0), garantindo que será enviado 1 aos bits de controle específicos apenas na instrução correta.

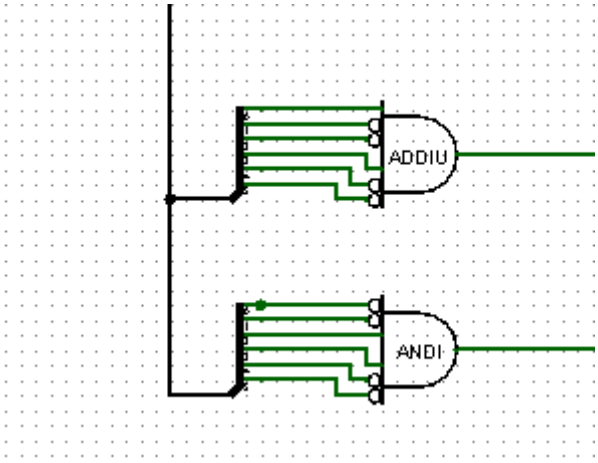
Consultando os OPCODEs da arquitetura MIPS, podemos constatar que ADDIU é representada por 001001 e ANDI por 001100.

MIPS Instruction Reference

Arithmetic and Logical Instructions

Instruction	Opcode/Function	Syntax	Operation
add	100000	f $d, $s, $t	$d = $s + $t
addu	100001	f $d, $s, $t	$d = $s + $t
addi	001000	f $d, $s, i	$d = $s + SE(i)
addiu	001001	f $d, $s, i	$d = $s + SE(i)
and	100100	f $d, $s, $t	$d = $s & $t
andi	001100	f $d, $s, i	$t = $s & ZE(i)

Com tudo isto estabelecido, podemos enfim montar as portas AND que receberão do bit 26 ao 31 do código da instrução:



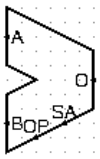
Dado os conjuntos de instruções utilizados, é evidente que em momento algum da execução os bits de controle MEM_TO_REG, STORE e LOAD serão utilizados (uma vez que não há qualquer tipo de manipulação de memória). Portanto, a fim de simplificar o circuito, podemos liga-los sempre no ‘terra’ (nível logico baixo).

Ambas instruções ativarão os sinais de controle REG_W (ou REG_WRITE), ALU_SRC e REG_DEST. O REG_W em 1 indica que o resultado volta pro registrador destino, o REG_DEST em 1 indica que é uma instrução de tipo I, ou seja, pegaremos os bits de 16 a 20 e, por fim, o ALU_SRC em 1 indica que a segunda entrada da ULA é do tipo imediato.

Vamos então enviar as saídas de ambas portas AND para um OR que ira ser conectado aos 3 sinais comentados, garantido que nos dois casos os bits de controle serão ativados.

Resta, então, a configuração do ALU_OP, que indica para a ULA qual tipo de operação será executada por ela utilizando as duas entradas recebidas. Neste caso, temos duas possíveis operações, um AND e uma soma, um ADD. Consultando a documentação da biblioteca de componentes ‘CS3410’, de onde vem a ULA utilizada, descobrimos os códigos para as operações desejadas:

MIPS ALU.



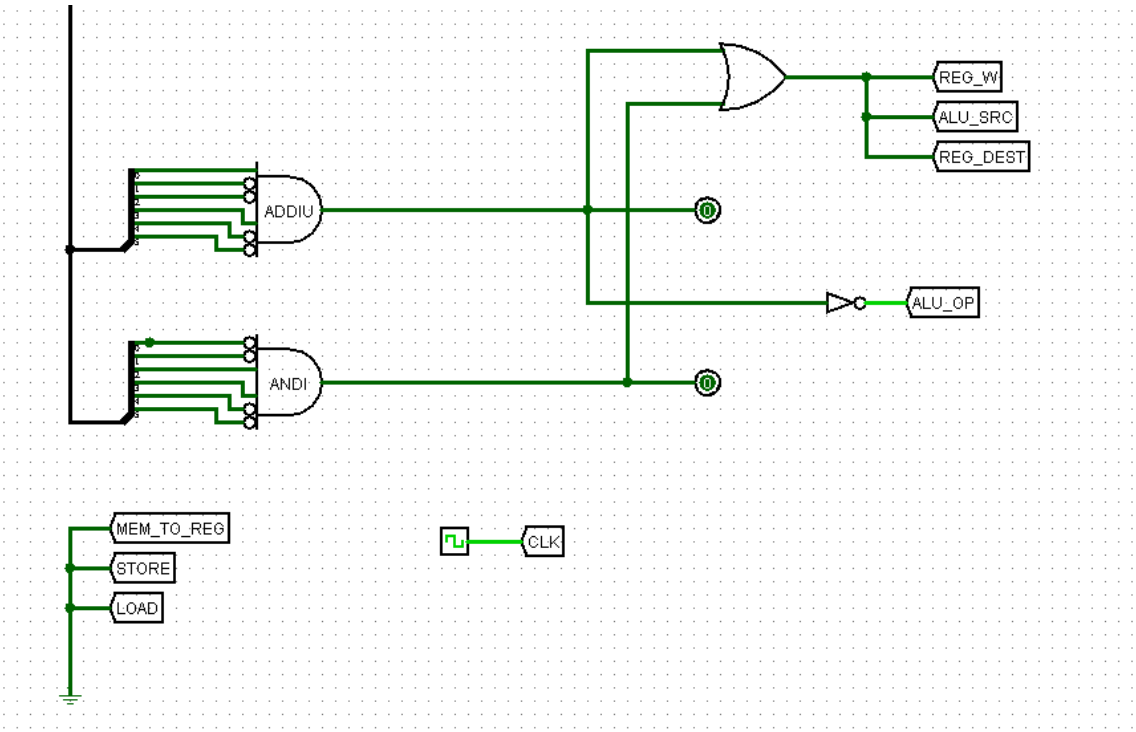
Computes a result as follows. You do not need to test the provided ALU, and can assume it will work exactly as specified.

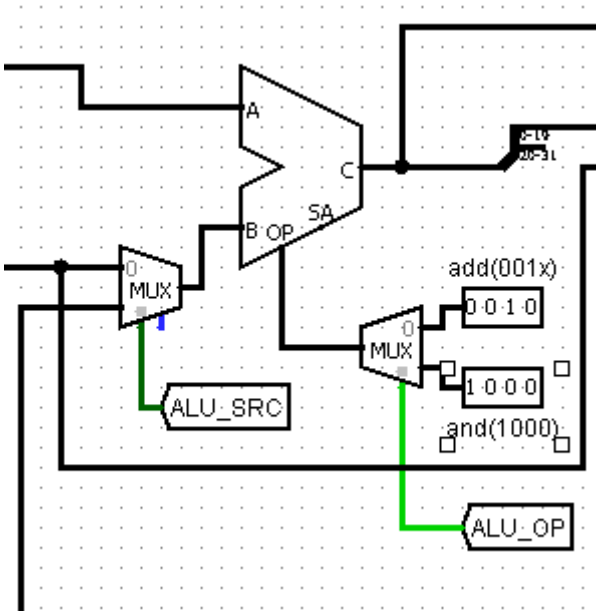
Op	name	C	V
000x	shift left logical	C = B << Sa	V = 0
001x	add	C = A + B	V = overflow
0100	shift right logical	C = B >>> Sa	V = 0
0101	shift right arithmetic	C = B >> Sa	V = 0
011x	subtract	C = A - B	V = overflow
1000	and	C = A & B	V = 0
1010	or	C = A | B	V = 0
1100	xor	C = A ^ B	V = 0
1110	nor	C = ~(A | B)	V = 0
1011	eq	C = (A == B) ? 000...0001 : 000...0000	V = 0
1001	ne	C = (A != B) ? 000...0001 : 000...0000	V = 0
1111	gt	C = (A > 0) ? 000...0001 : 000...0000	V = 0
1101	le	C = (A ≤ 0) ? 000...0001 : 000...0000	V = 0

Porém, há apenas 1 bit sendo enviado para o ALUP, 1 ou 0. Para que não haja necessidade de usar o FUNCT, complicando o circuito, pode-se apenas usar um MUX para decidir qual das duas operações é a correta para a instrução sendo executada, com o ALU_OP sendo o bit de seleção.

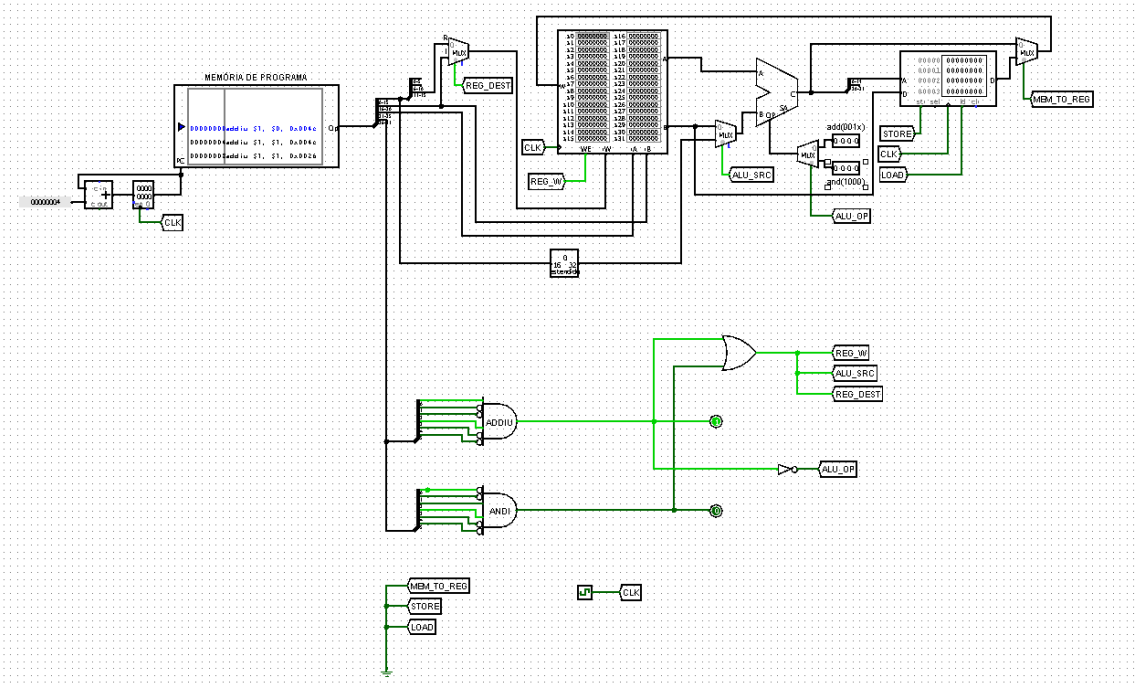
Uma forma simples de fazer isso, é conectando o ALU_OP apenas a uma das portas AND e negando. Fazendo com que o 0 seja a operação necessária para aquela instrução e 1 para a outra. Basta apenas setar corretamente o código de cada instrução na porta correta do MUX.

Finalmente, o circuito logico completo fica dessa forma:





Sendo o **circuito completo**:



E as **instruções** são:

```
addiu $1, $0, 76
addiu $1, $1, 76
addiu $1, $1, 38
addiu $1, $1, 38
andi $2, $1, 255
```

O resultado final estará armazenado no registrador temporário $2.