Problema 2

A) Controle Monociclo: considere o controle monociclo visto em aula e o conjunto de instruções colocado como solução para o problema 1. Quais alterações são necessárias em tal controle monociclo para ser acrescentada a instrução "Add $r1, 8($r2)" que soma o conteúdo do registrador $r1 com o conteúdo da memória na posição 8+$r2 e salva na mesma posição, isto é, Mem[$r2+8] = $r1 + Mem[$r2+8].

Conjunto de Instruções

- **Instrução** **Significado**
  `add $r1, 8($r2)` `Mem[$r2+8] = Mem[$r2+8] + $r1`

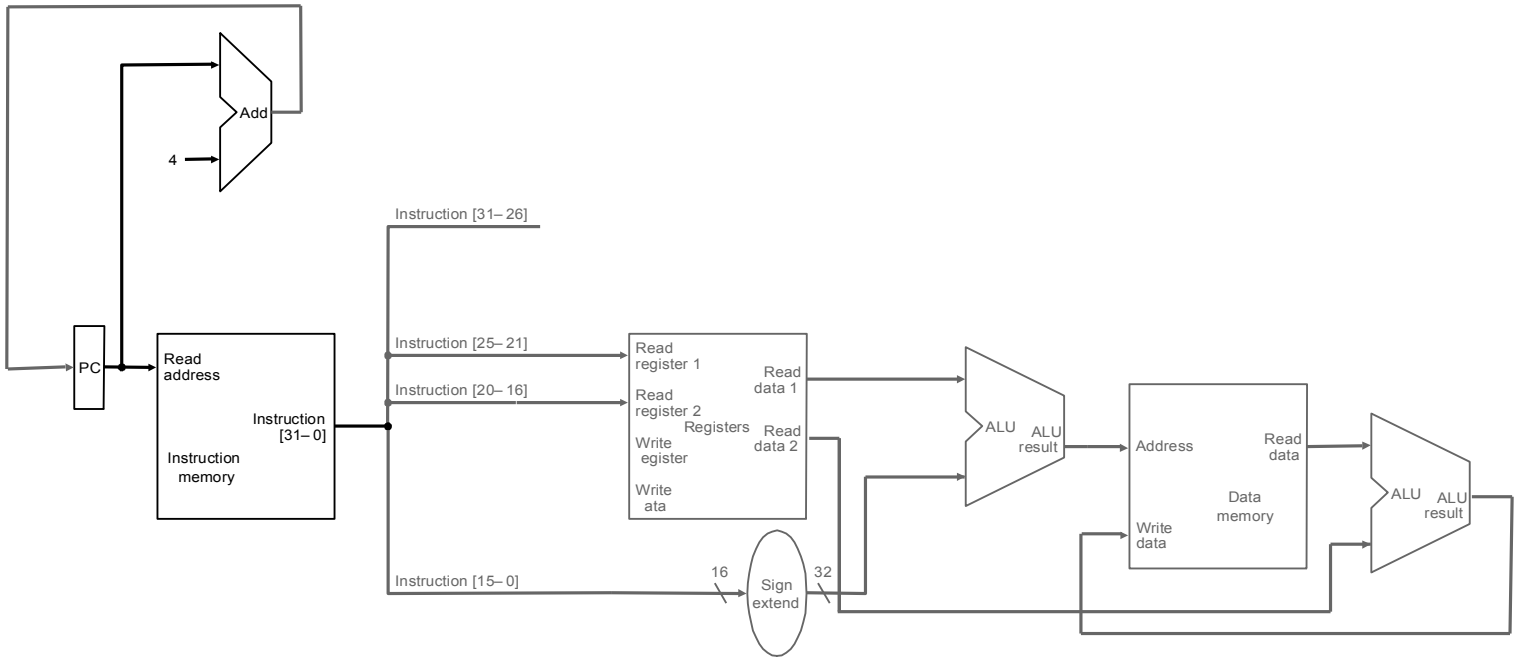
- **Formatos:**

op (6 bits)	r2 (5 bits)	r1 (5 bits)	offset (16 bits)

- **Exemplo: add $5, 8($7)**

101010	00111	00101	0000000000001000

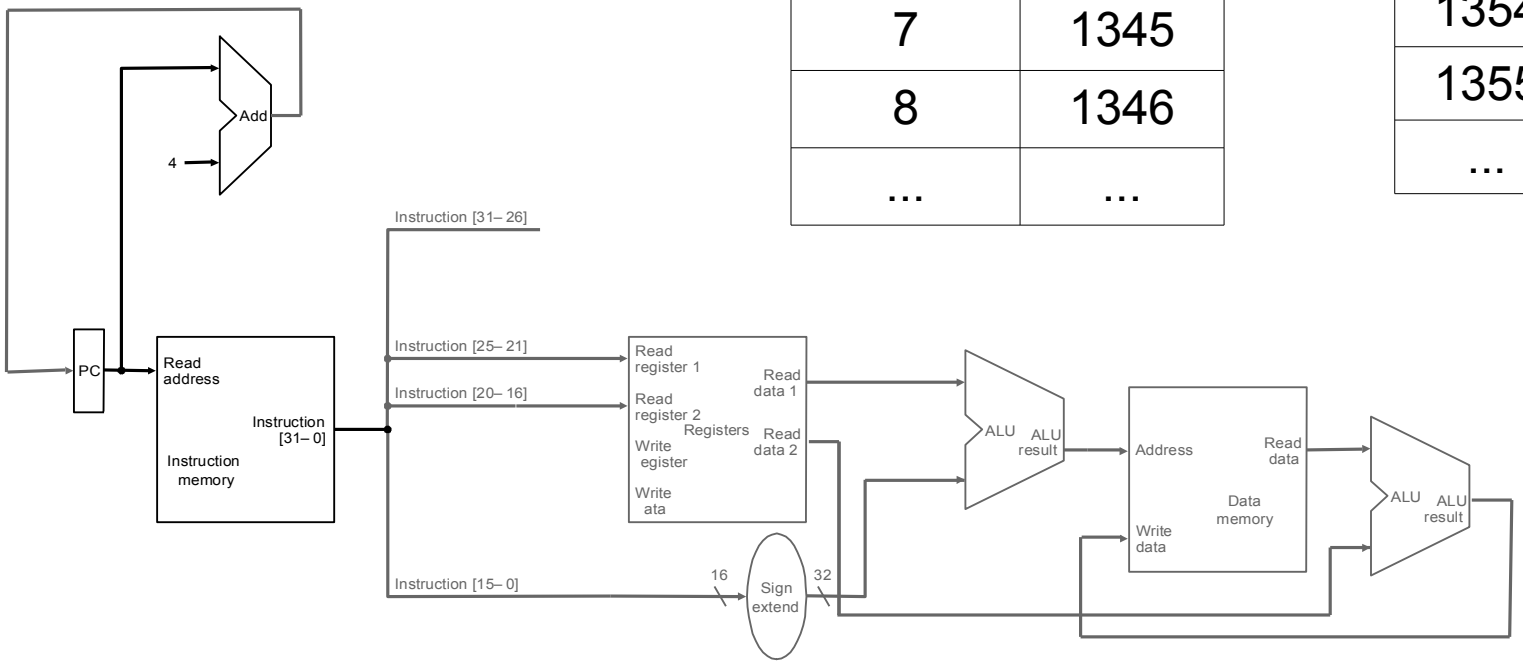
Caminho de Dados: add $5, 8($7)



Execução: add $5, 8($7)

Registradores	
end.	valor
...	...
4	345
5	350
6	347
7	1345
8	1346
...	...

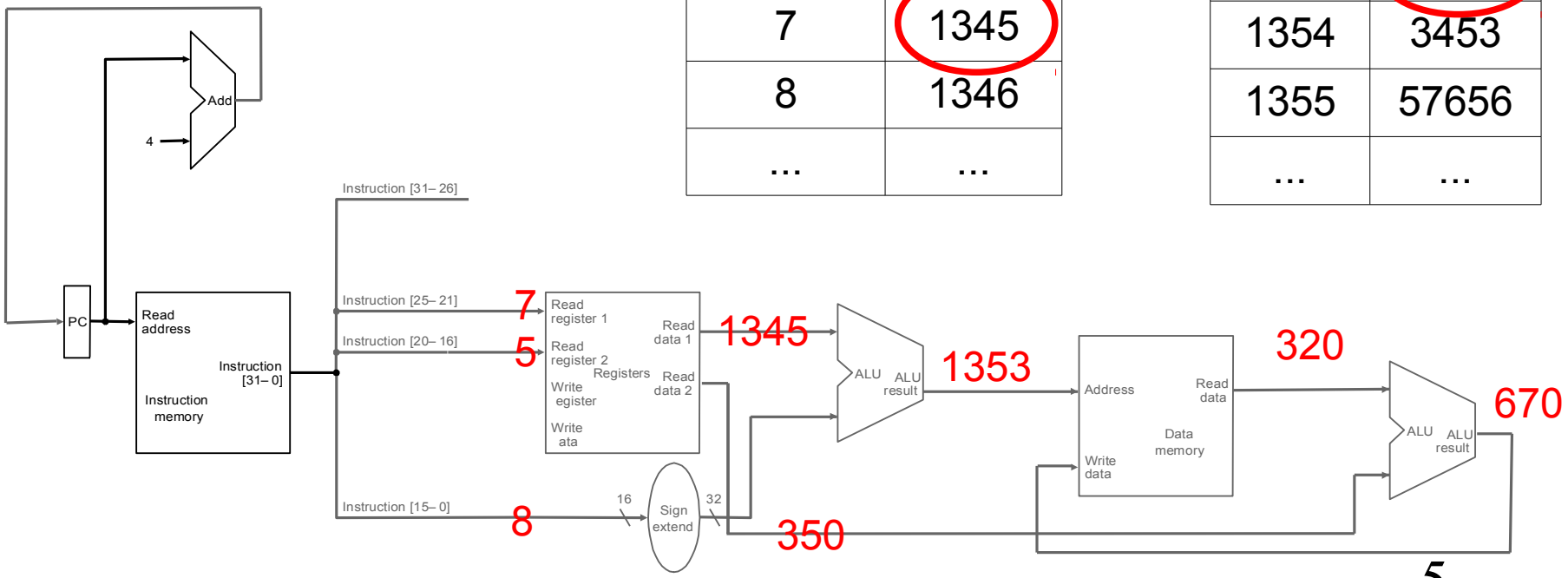
Memória	
end.	valor
...	...
1351	345
1352	122
1353	320
1354	3453
1355	57656
...	...



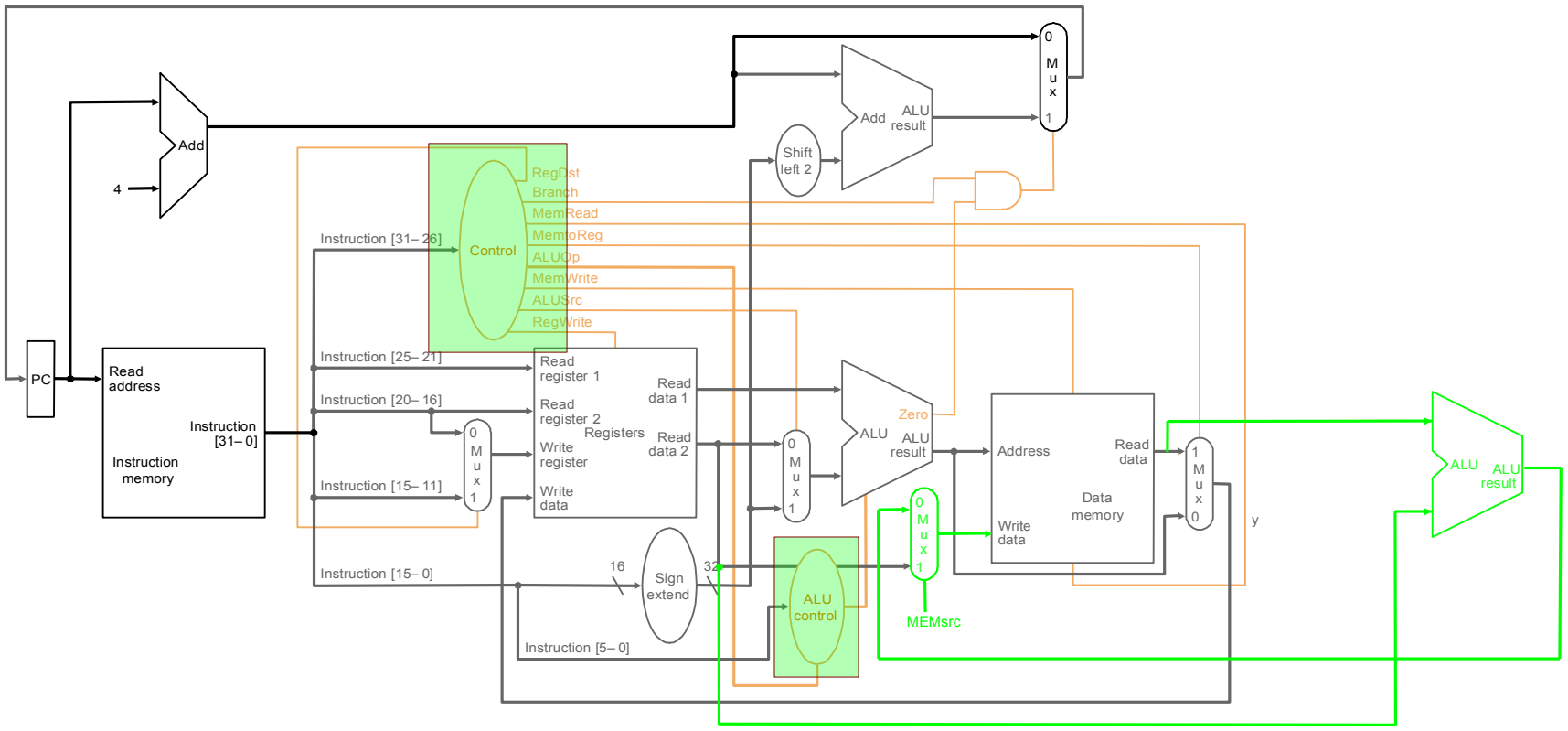
Execução: add $5, 8($7)

Registradores	
end.	valor
...	...
4	345
5	350
6	347
7	1345
8	1346
...	...

Memória	
end.	valor
...	...
1351	345
1352	122
1353	320
1354	3453
1355	57656
...	...



Integração


Add
4
PC
Read address
Instruction [31– 0]
Instruction memory
Instruction [31– 26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25– 21]
Instruction [20– 16]
Instruction [15– 11]
Instruction [15– 0]
Instruction [5– 0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
0 M u x 1
16
Sign extend
32
Shift left 2
Add
ALU result
ALU
Zero
ALU result
ALU control
Address
Read data
Data memory
Write data
MEMsrc
y
ALU
ALU result

Problema 2

B) Controle Multiclo: considerando novamente o problema 1, especifique uma implementação multiciclo para a instrução "Add 6($r1), 8($r2)" que significa Mem[$r2+8] = Mem[$r1+6] + Mem[$r2+8].

Conjunto de Instruções

- **Instrução** **Significado**
  `add 6($r1), 8($r2)` `Mem[$r2+8] = Mem[$r1+6] + Mem[$r2+8]`

- **Formatos:**

op (6 bits)	r2 (5 bits)	r1 (5 bits)	offset 2 (8 bits)	offset 1 (8 bits)

- **Exemplo: add 8($5), 4($7)**

010101	00101	00111	00001000	00000110

Ciclos das Instruções MIPs

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR = Memory[PC] PC = PC + 4			
Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A ==B) then PC = ALUOut	PC = PC [31-28] II (IR[25-0]<<2)
Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

Solução com 8 ciclos

- **Instrução** **Significado**
  `add 6($r1), 8($r2)` `Mem[$r2+8] = Mem[$r1+6] + Mem[$r2+8]`
- **Formatos:**

op (6 bits)	r2 (5 bits)	r1 (5 bits)	offset 2 (8 bits)	offset 1 (8 bits)

- **Ciclos:**
  - **IR = Memory[PC]; PC = PC+4 (IR guarda instrução)**
  - **A = Reg[IR[20-16]]; B = Reg[IR[25-21]] (A guarda o valor do registrador r1 e B guarda o valor do registrador r2)**
  - **E1 = A + sign-extend(IR[7-0]) (E1 guarda o endereço do primeiro operando)**
  - **E2 = B + sign-extend(IR[15-8]) (E2 guarda o endereço do segundo operando)**
  - **A = Memory[E1] (A guarda o valor do primeiro operando)**
  - **B = Memory[E2] (B guarda o valor do segundo operando)**
  - **ALUout = A + B (ALUout guarda o resultado da operação)**
  - **Memory[E2] = ALUout (Resultado da operação está na memória)**

Solução com 7 ciclos

- **Instrução** **Significado**
  `add 6($r1), 8($r2)` `Mem[$r2+8] = Mem[$r1+6] + Mem[$r2+8]`
- **Formatos:**

op (6 bits)	r2 (5 bits)	r1 (5 bits)	offset 2 (8 bits)	offset 1 (8 bits)

- **Ciclos:**
  - **IR = Memory[PC]; PC = PC+4 (IR guarda instrução)**
  - **A = Reg[IR[20-16]]; B = Reg[IR[25-21]]**
  - **E1 = A + sign-extend(IR[15-8])**
  - **E2 = B + sign-extend(IR[7-0]); A = Memory[E1]**
  - **B = Memory[E2]**
  - **ALUout = A + B**
  - **Memory[E2] = ALUout**