Sistemi a Microcontrollore

2E. Bus e Mappaggio in Memoria

Anno Accademico 2019/2020

Indice

- Architettura con Bus e Mappaggio in Memoria
- Modifiche all'Architettura
  - Mappaggio della Memoria Dati
  - Mappaggio dell'I/O Parallelo
  - Rotazione dell'Immediate
- Esercitazione

Indice

- Architettura con Bus e Mappaggio in Memoria
- Modifiche all'Architettura
  - Mappaggio della Memoria Dati
  - Mappaggio dell'I/O Parallelo
  - Rotazione dell'Immediate
- Esercitazione

Architettura con Bus e Mappaggio in Memoria

- L'architettura semplificata di **ARM LEGv8** vista finora è in realtà un'architettura di processore generico
- Per rendere tale sistema **un'architettura di microcontrollore** occorre **dotarla di un bus** che possa connettere il processore alle **periferiche**
- L'**architettura di riferimento** per rendere il LEGv8 un microcontrollore è quella dell'**ARMv6-M** (famiglia dedicata ai microcontrollori)

Architettura con Bus e Mappaggio in Memoria

- Rispetto all'ISA ARMv6-M, il sistema di bus e mappaggio in memoria adottato ha delle semplificazioni
  - in partenza possiede **solo una porta di I/O parallelo** (GPIOA) e la **memoria dati** (dmem) mappate in memoria
  - utilizza **un solo registro di configurazione** (MODER) per le porte di I/O parallelo, assieme ai registri di dato (IDR e ODR)
  - prevede **due sole modalità di I/O** per le porte: input (MODERy[1:0]=2'b00) e output (MODERy[1:0]=2'b01) digitale

Architettura con Bus e Mappaggio in Memoria

- Rispetto all'ISA ARMv6-M, il sistema di bus e mappaggio in memoria adottato ha delle semplificazioni

boundary address	size	memory area/peripheral
0x20000000 – 0x20001FFF	8 kB	SRAM
0x48000000 – 0x480003FF	1 kB	GPIOA

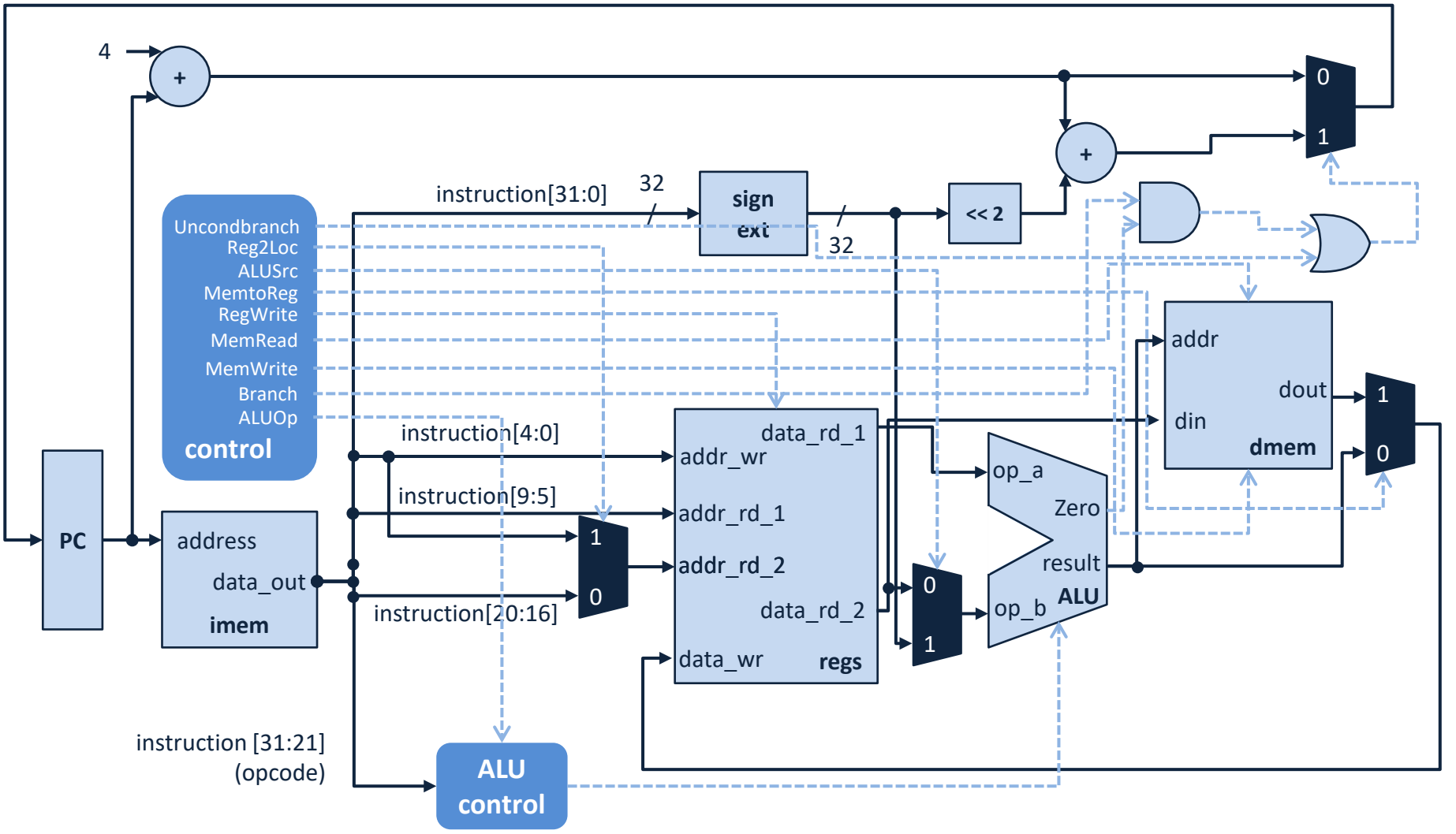
offset	register	content			
0x00	GPIOx_MODER	MODER15[1:0]	...	MODER1[1:0]	MODER0[1:0]
0x10	GPIO_x_IDR			ID15:0	
0x14	GPIO_x_ODR			OD15:0	

MODERy[1:0]	I/O pin configuration
01	GP output
00	input

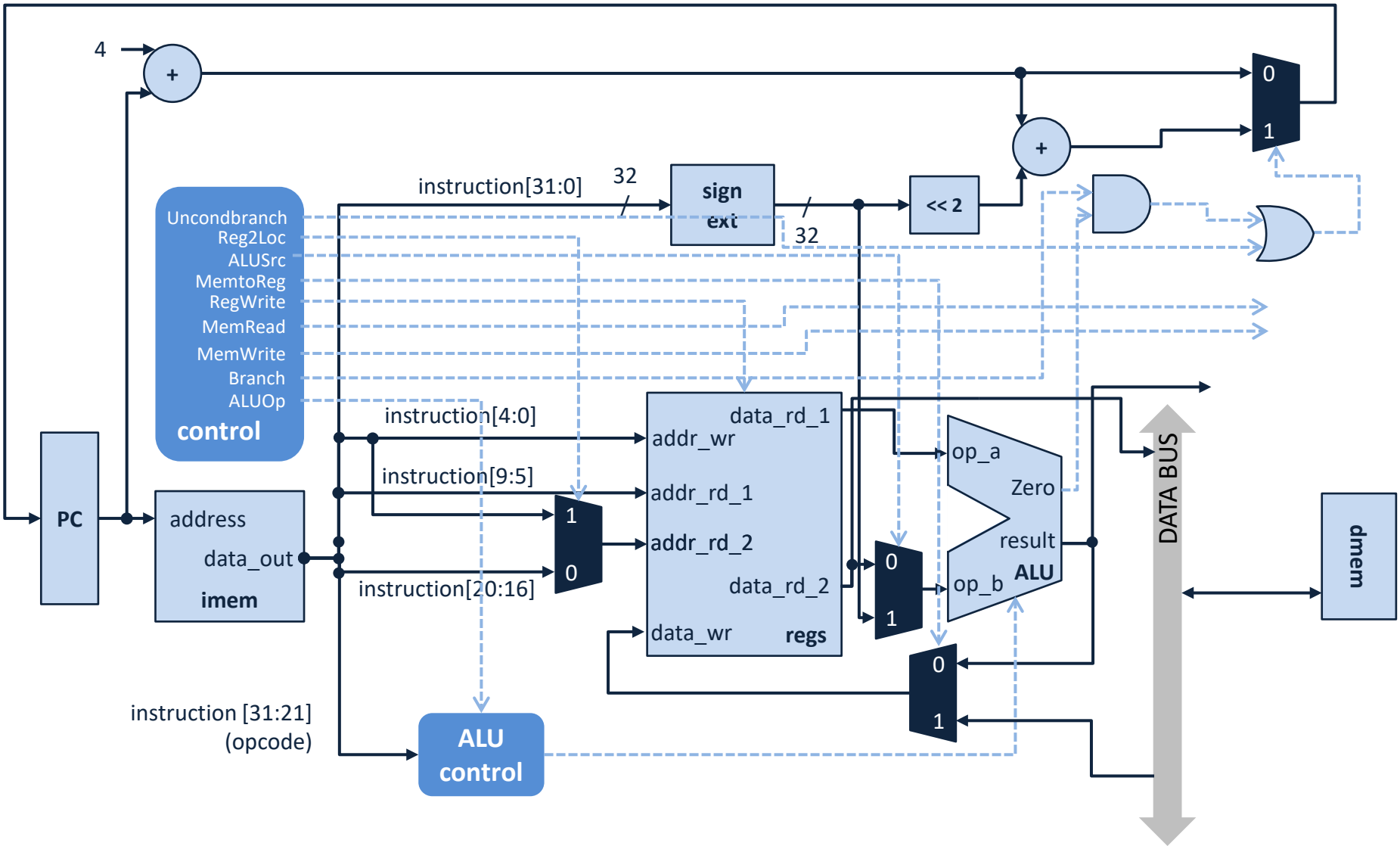
Indice

- Architettura con Bus e Mappaggio in Memoria
- Modifiche all'Architettura
  - Mappaggio della Memoria Dati
  - Mappaggio dell'I/O Parallelo
  - Rotazione dell'Immediate
- Esercitazione

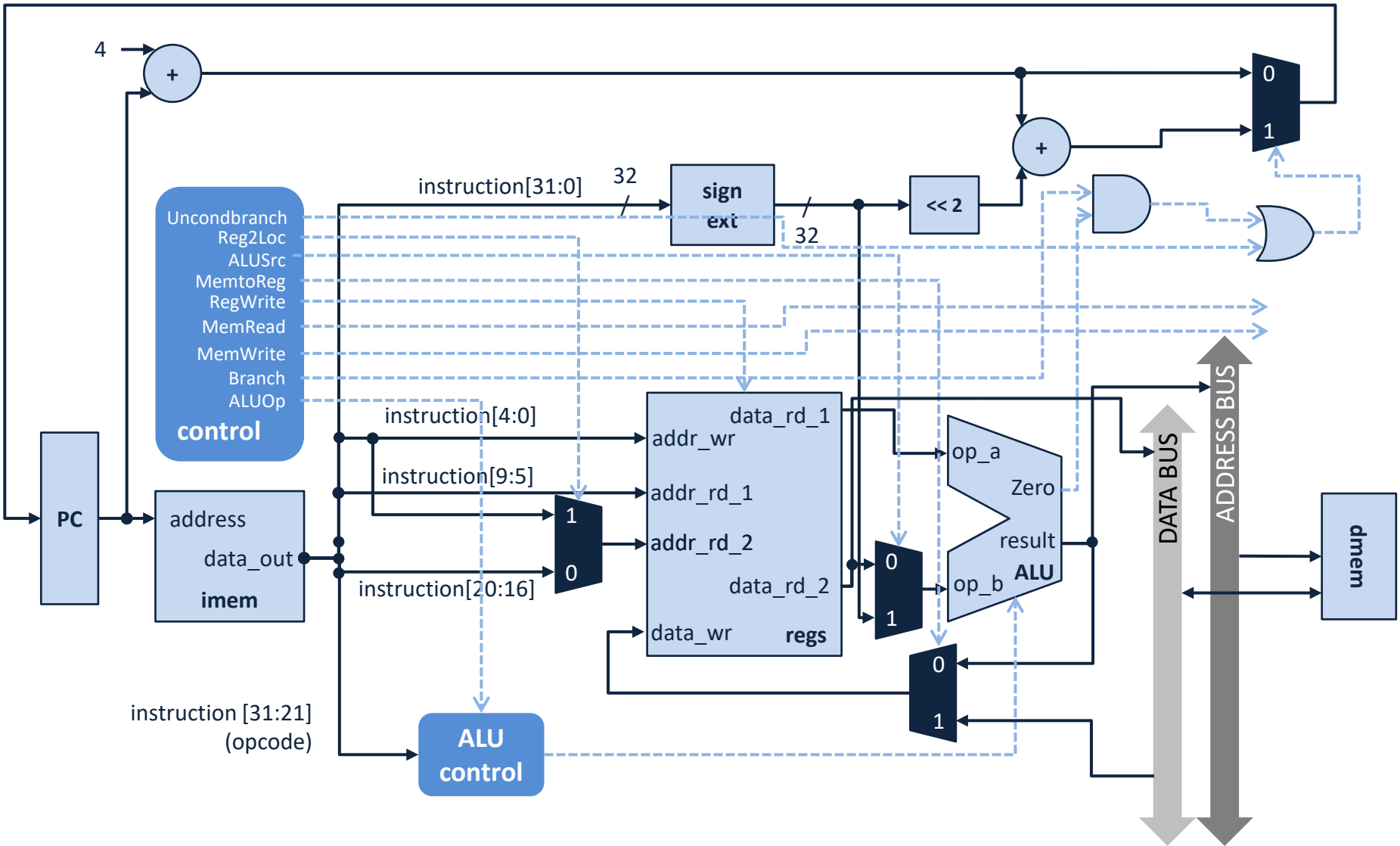
Mappaggio della Memoria Dati



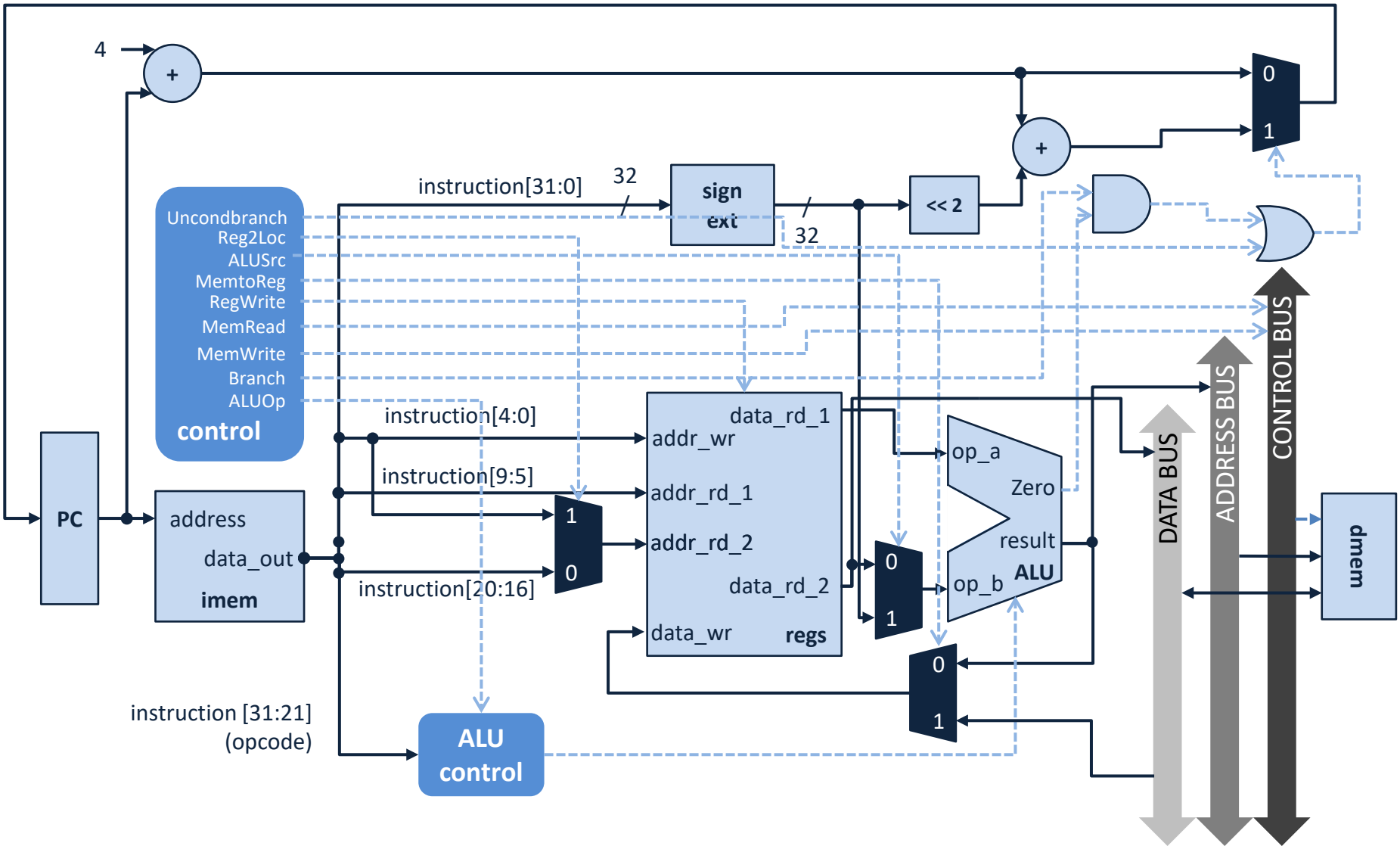
Mappaggio della Memoria Dati



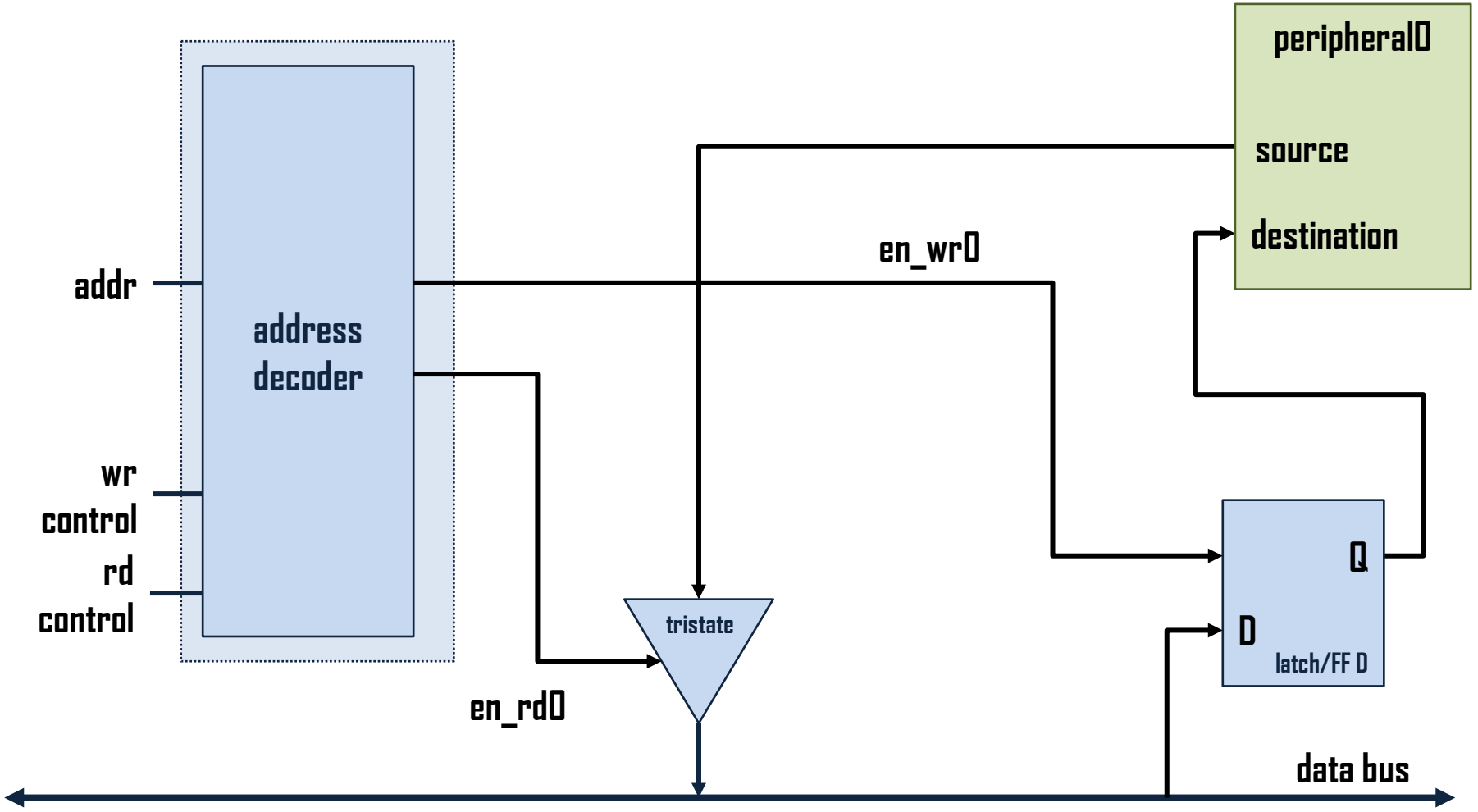
Mappaggio della Memoria Dati



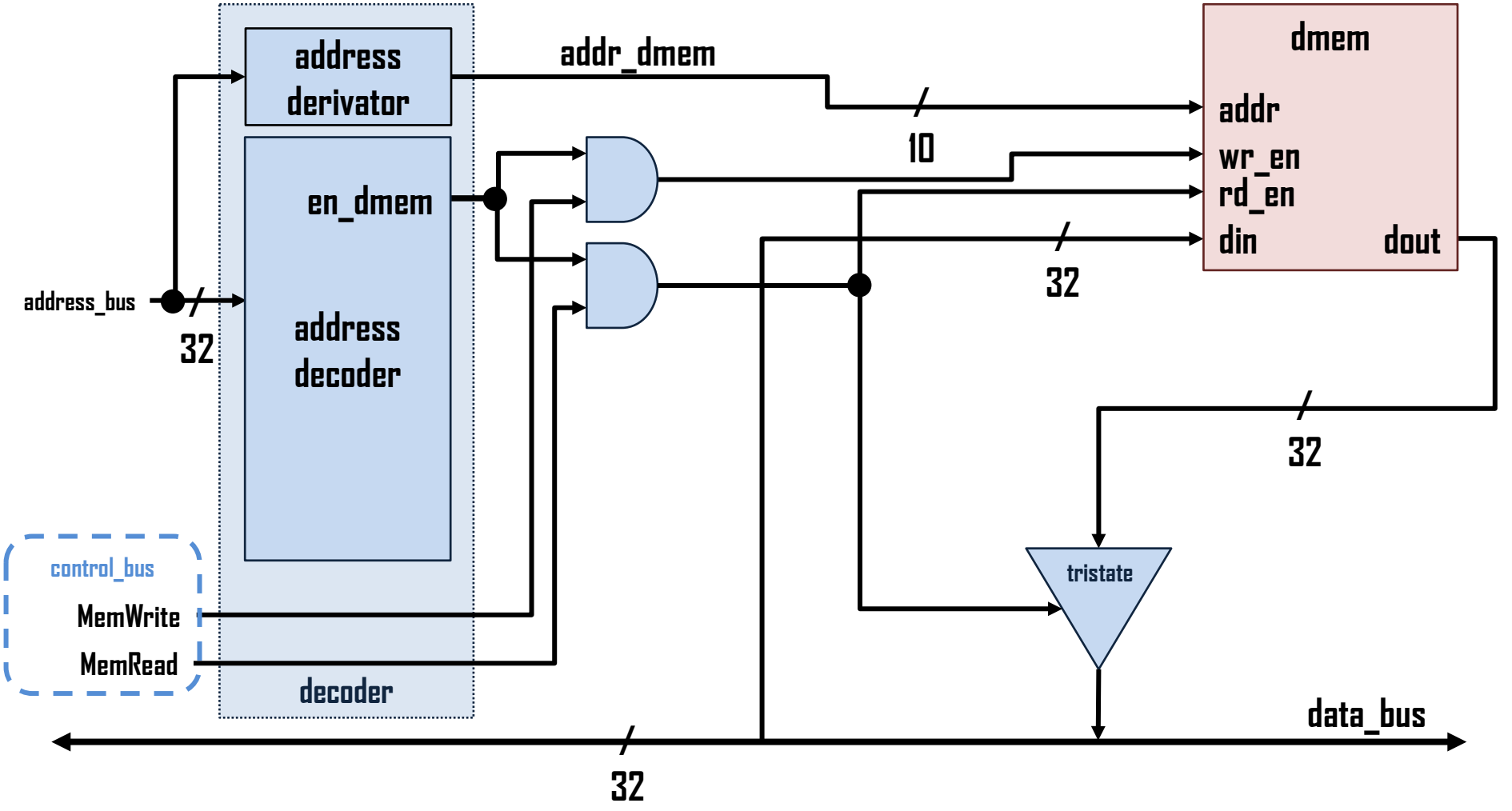
Mappaggio della Memoria Dati



Mappaggio della Memoria Dati



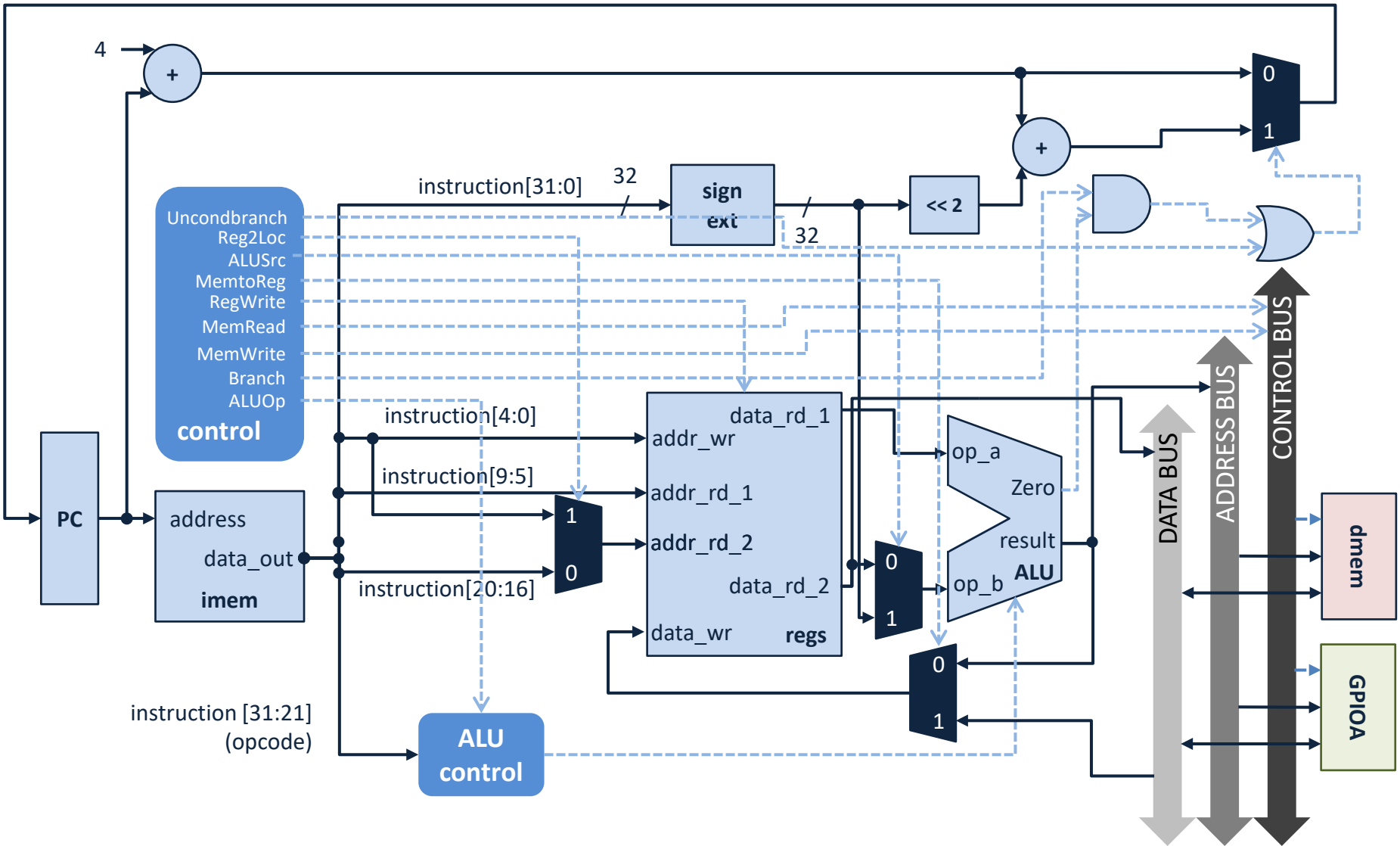
Mappaggio della Memoria Dati



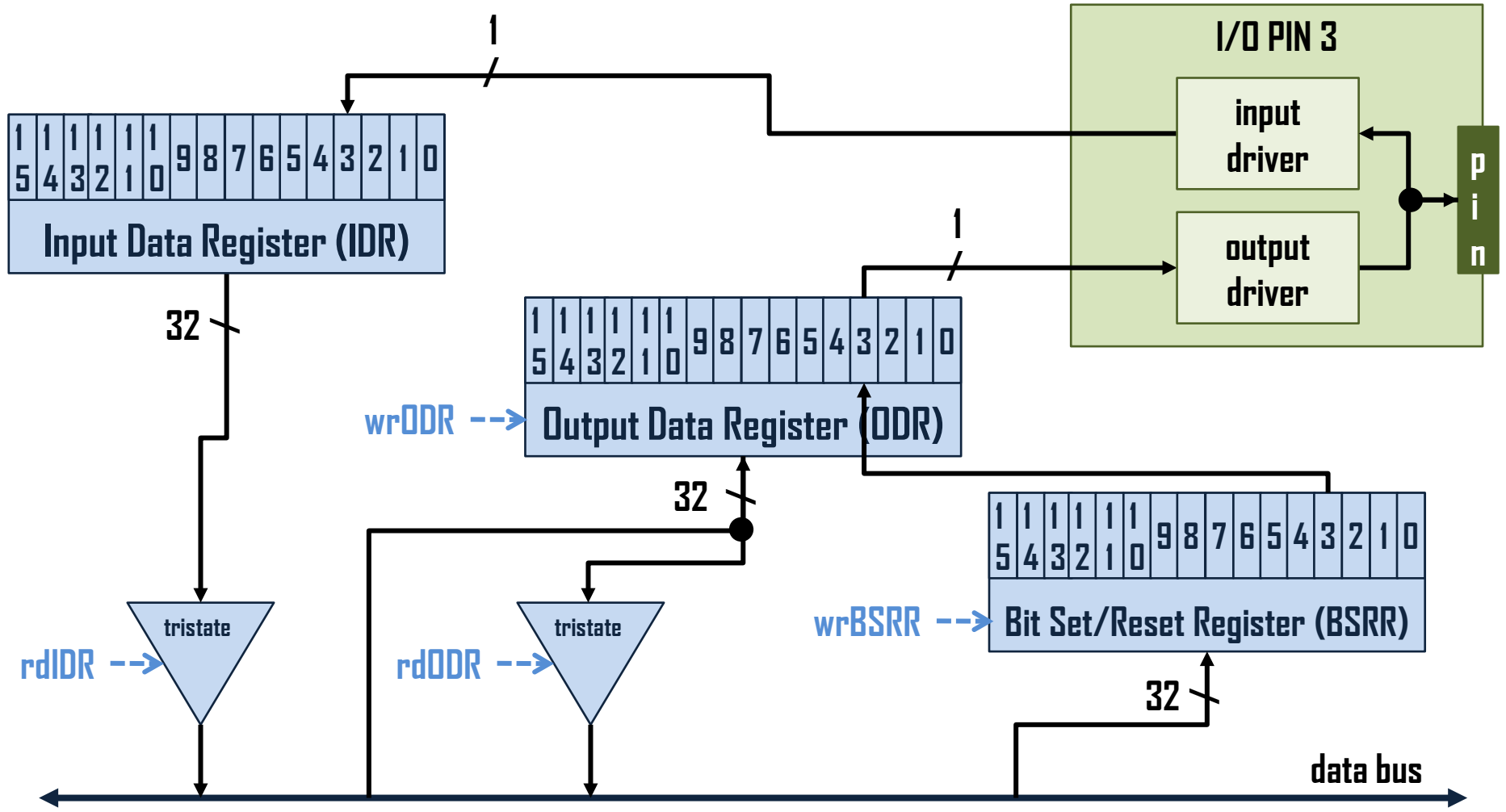
Indice

- Architettura con Bus e Mappaggio in Memoria
- Modifiche all'Architettura
  - Mappaggio della Memoria Dati
  - Mappaggio dell'I/O Parallelo
  - Rotazione dell'Immediate
- Esercitazione

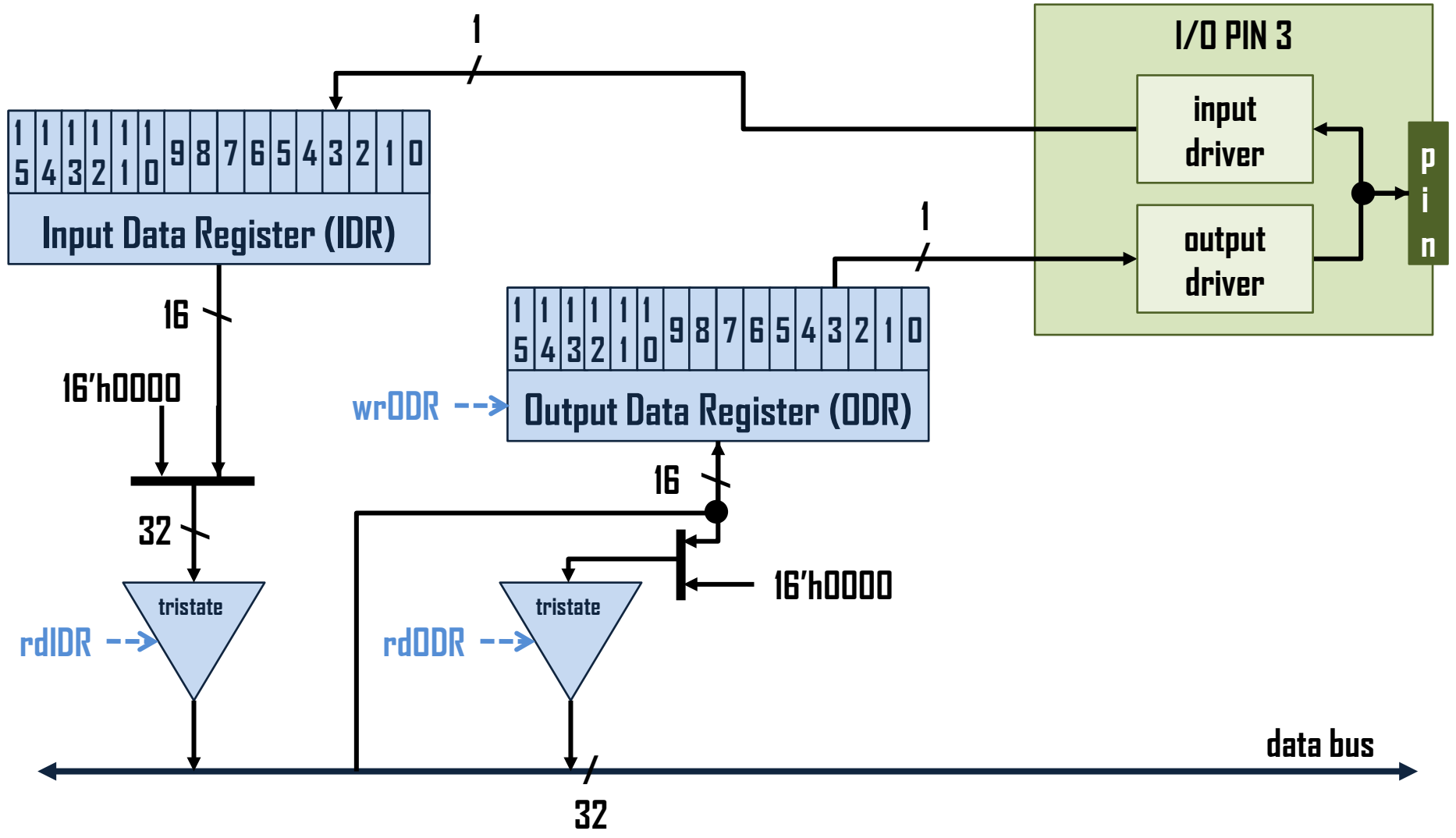
Mappaggio dell'I/O Parallelo



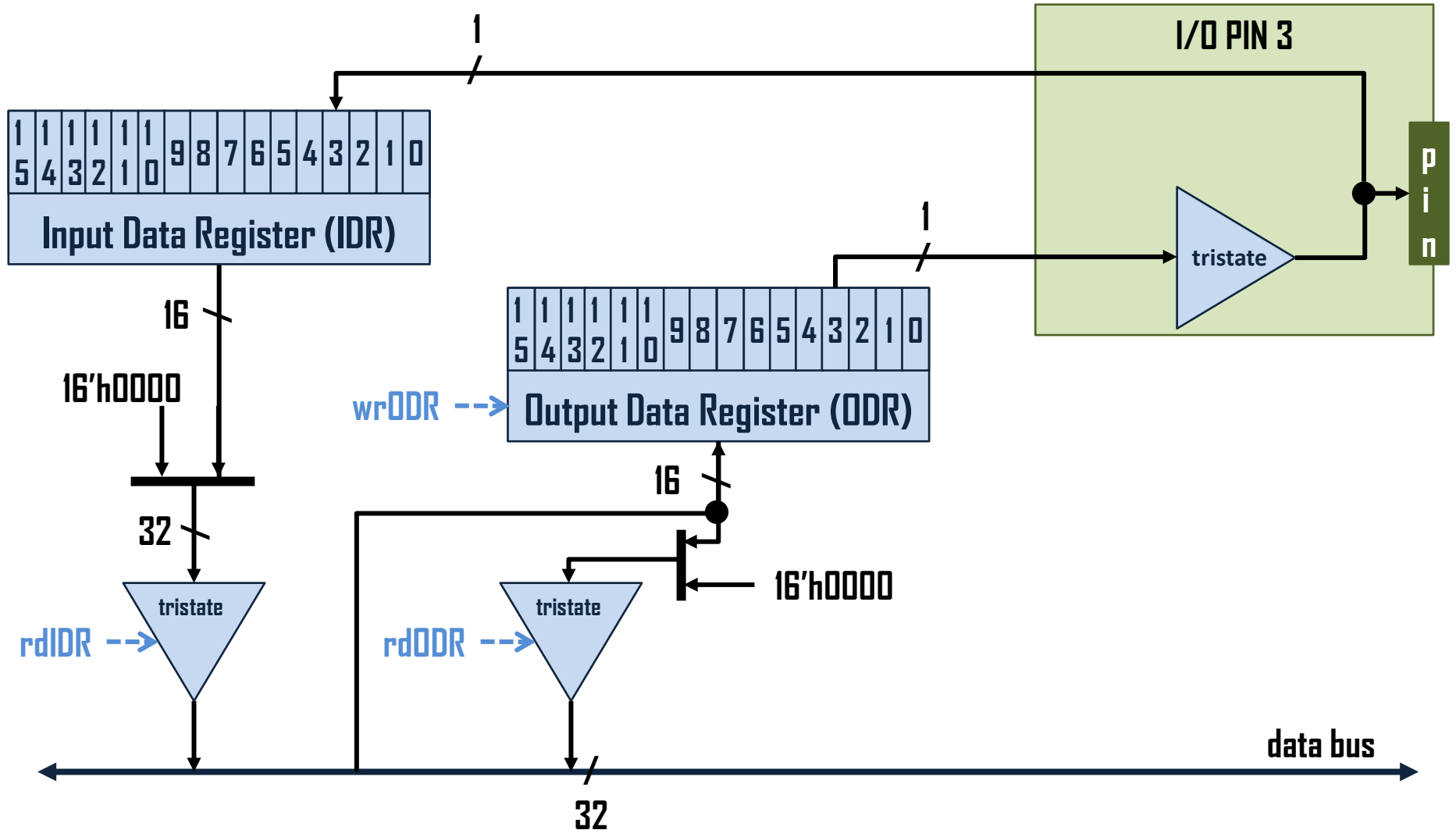
Mappaggio dell'I/O Parallelo



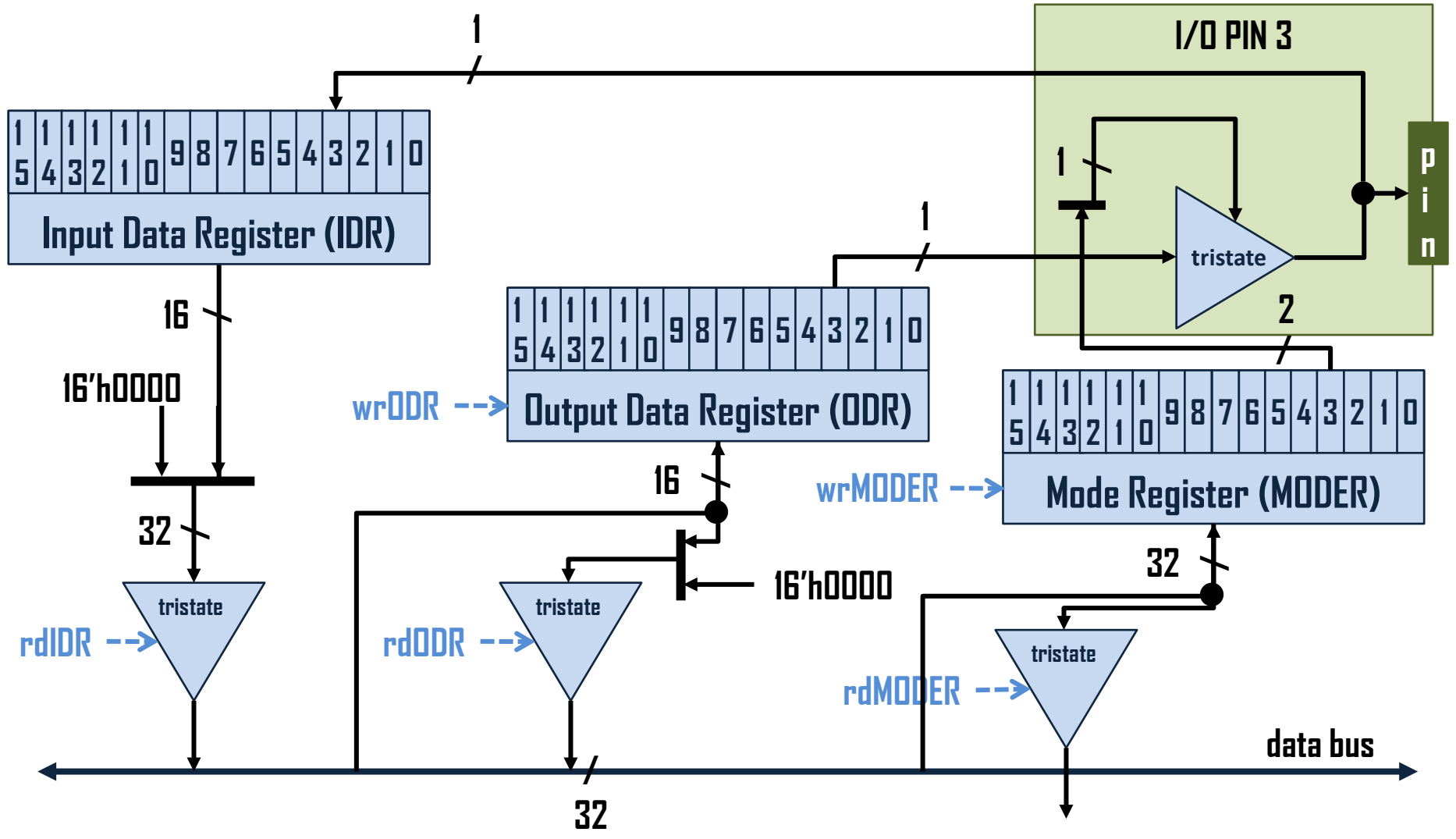
Mappaggio dell'I/O Parallelo



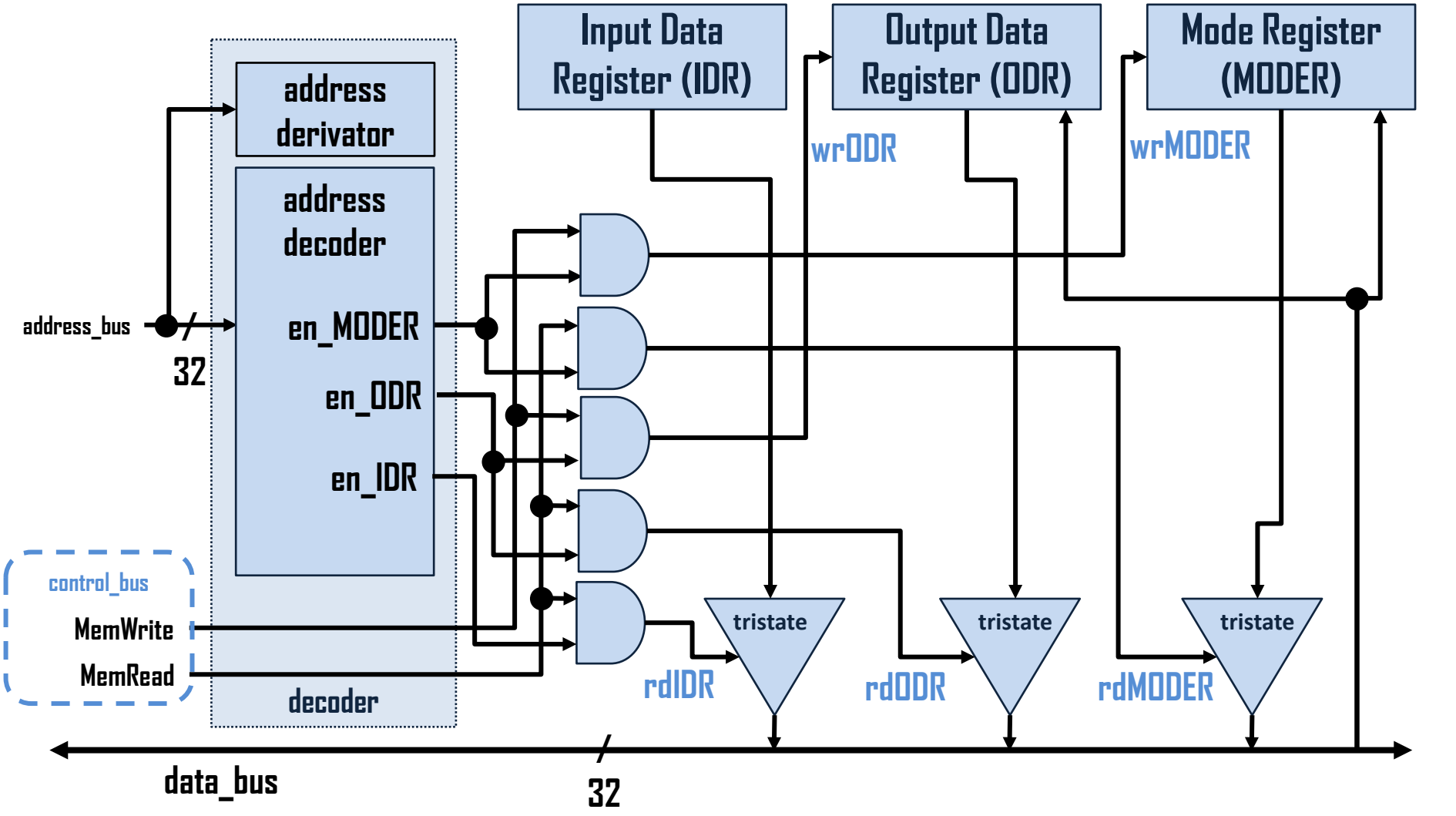
Mappaggio dell'I/O Parallelo



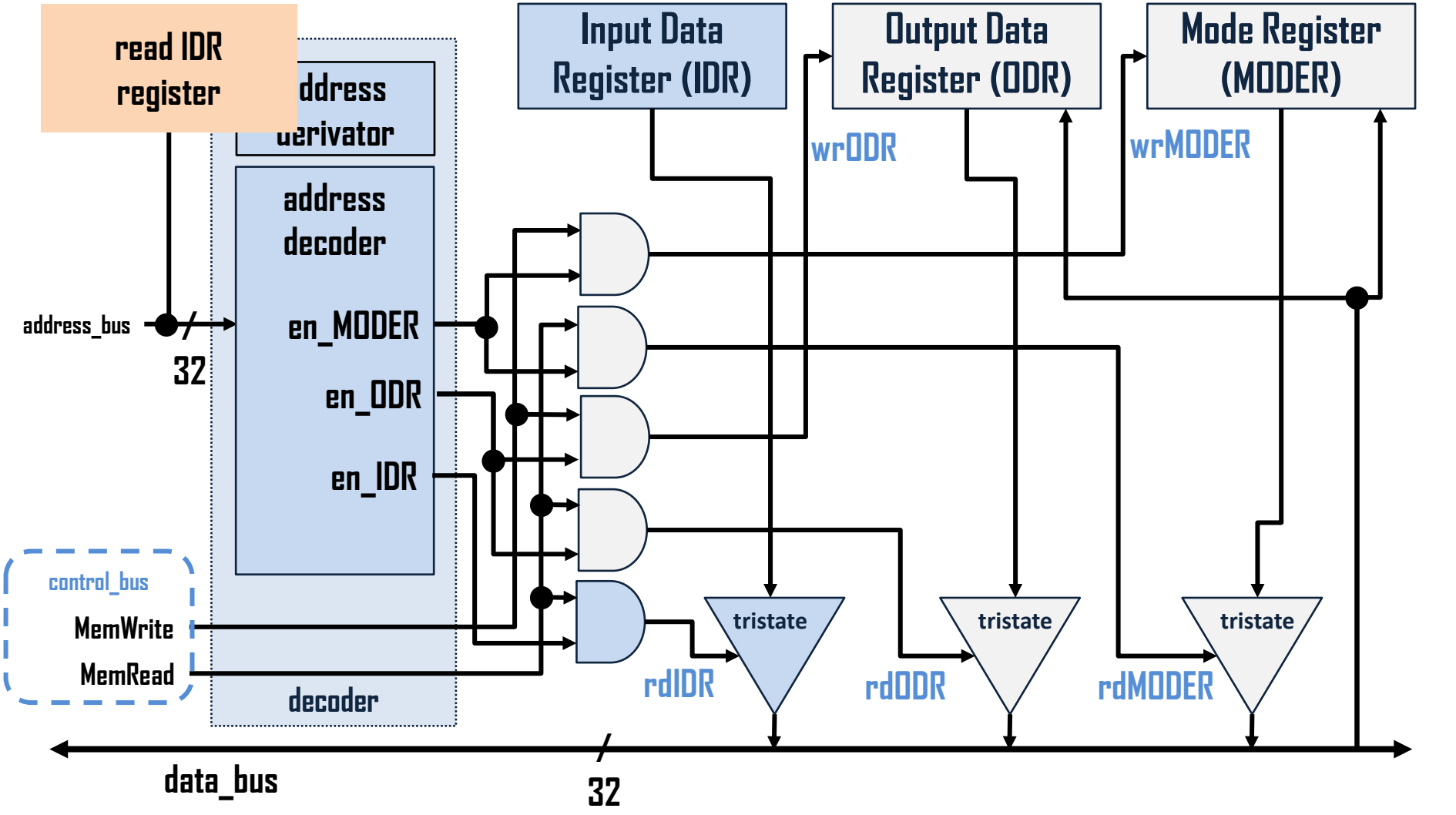
Mappaggio dell'I/O Parallelo



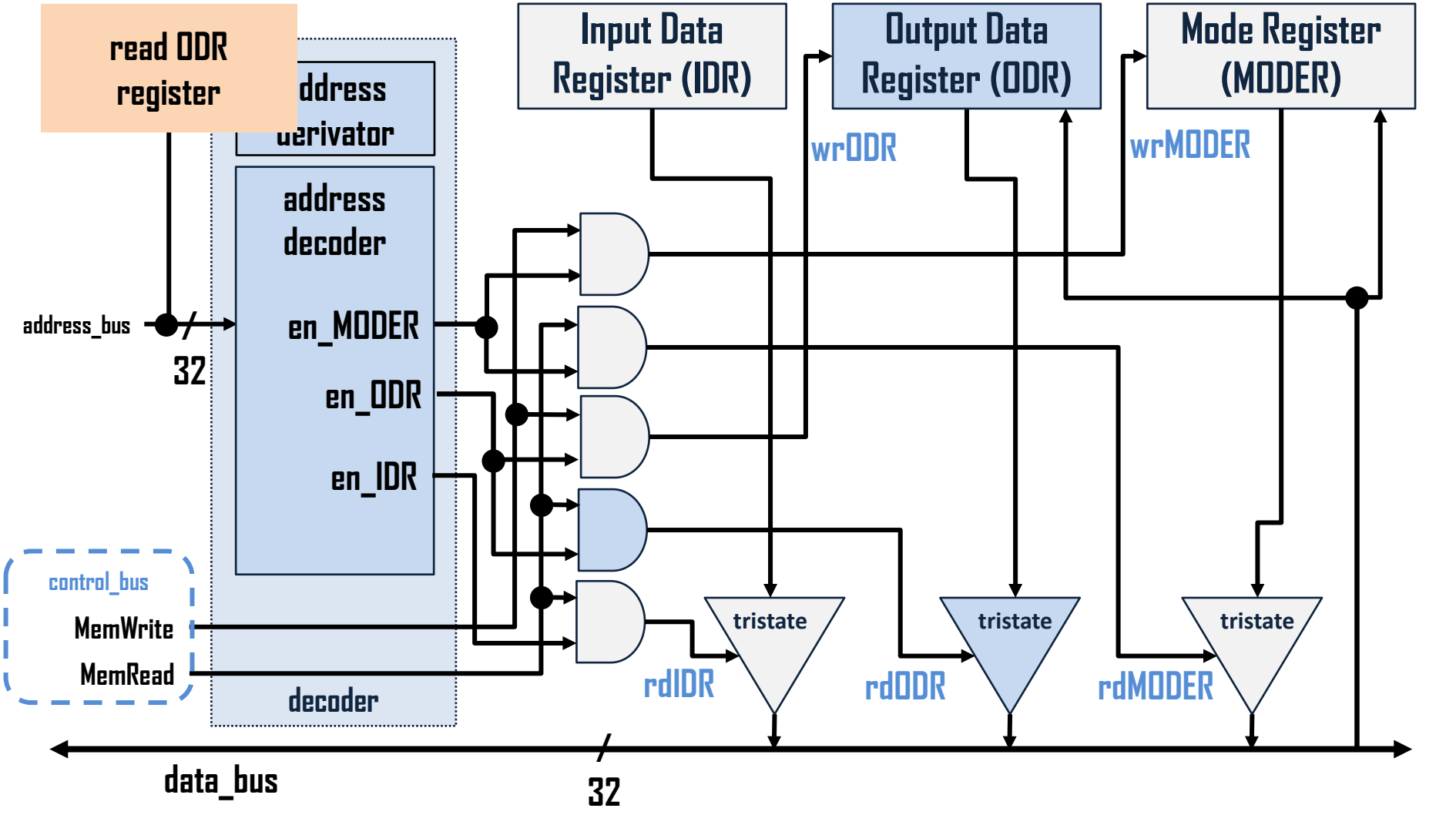
Mappaggio dell'I/O Parallelo



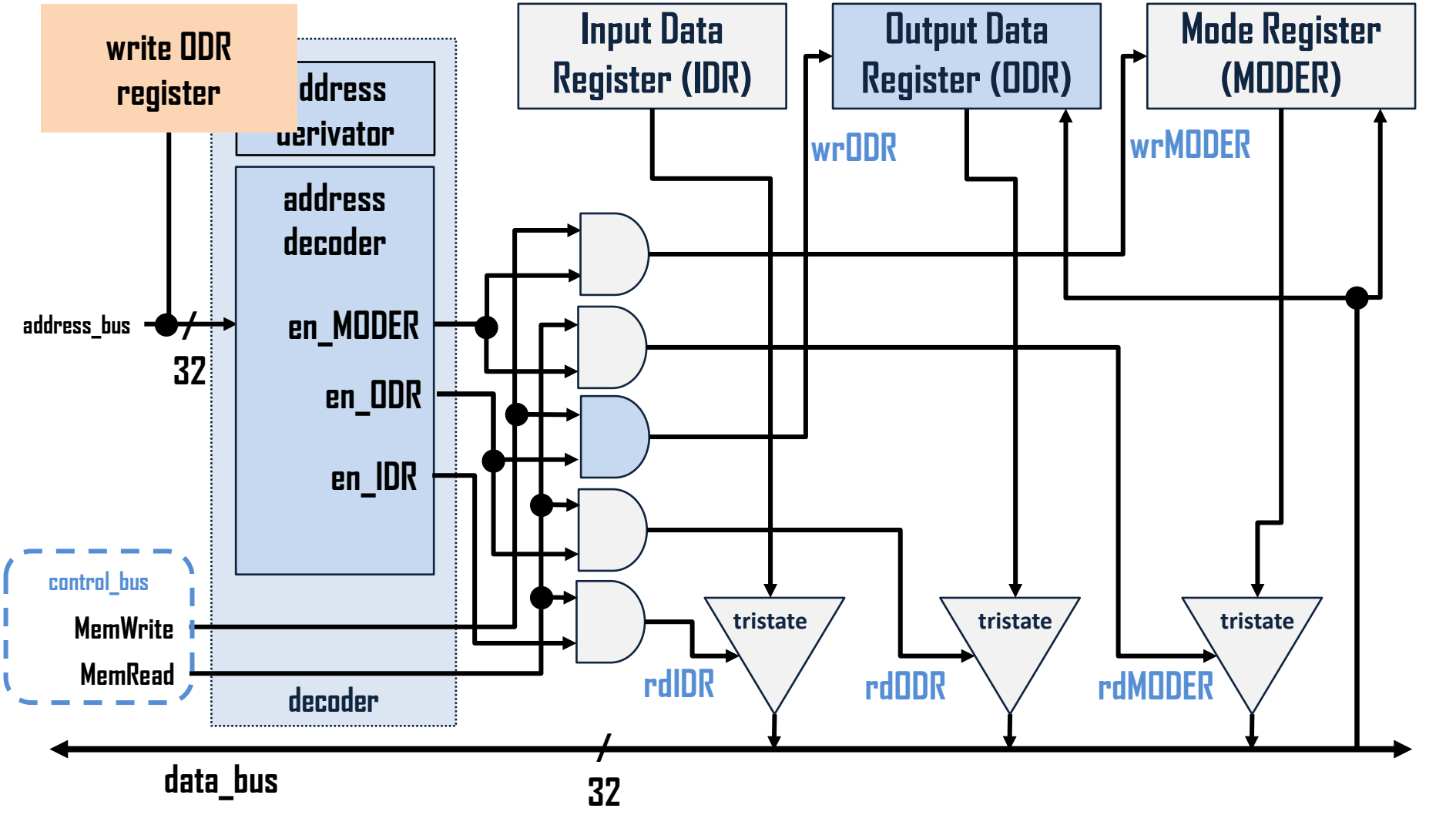
Mappaggio dell'I/O Parallelo



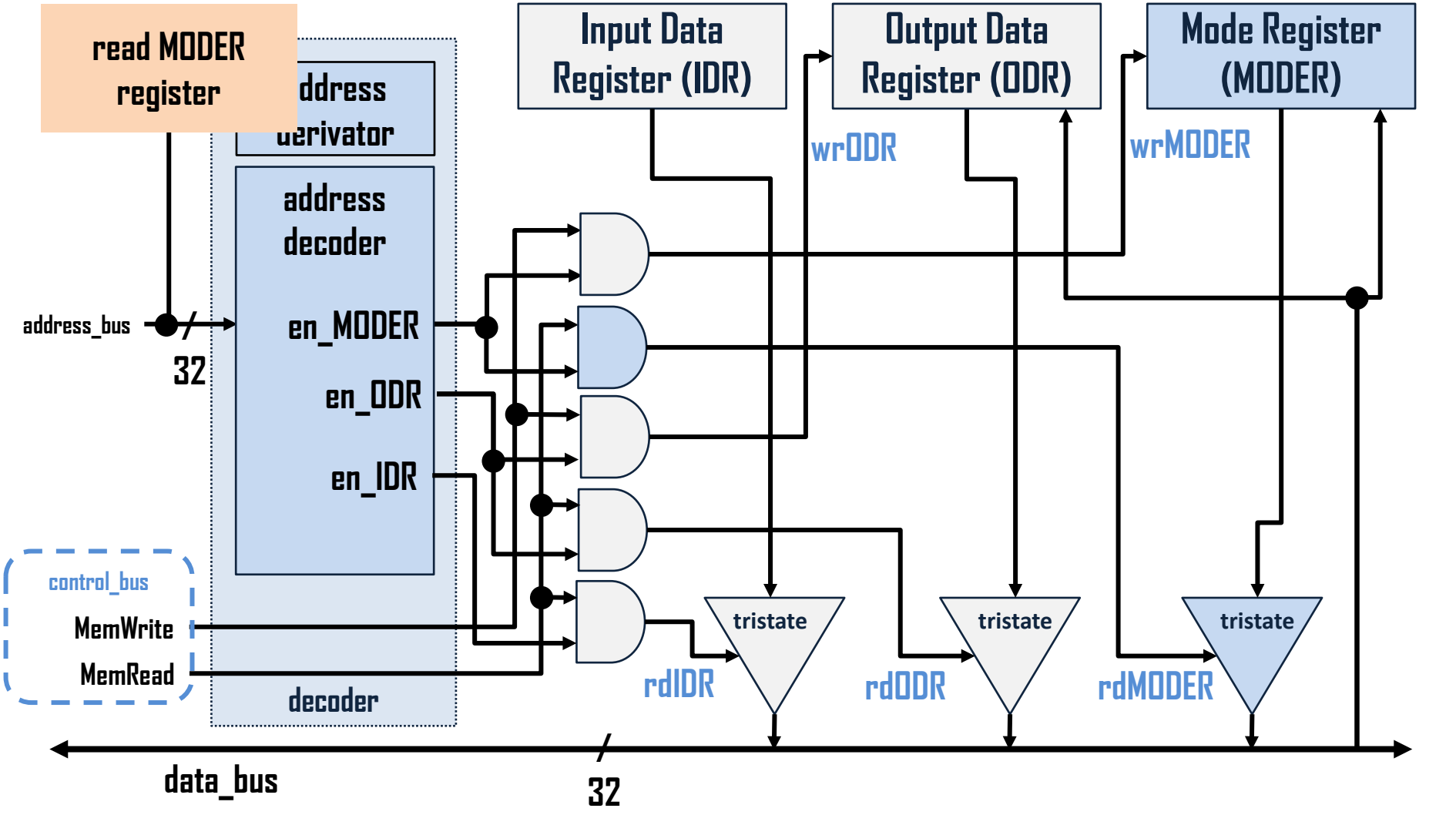
Mappaggio dell'I/O Parallelo



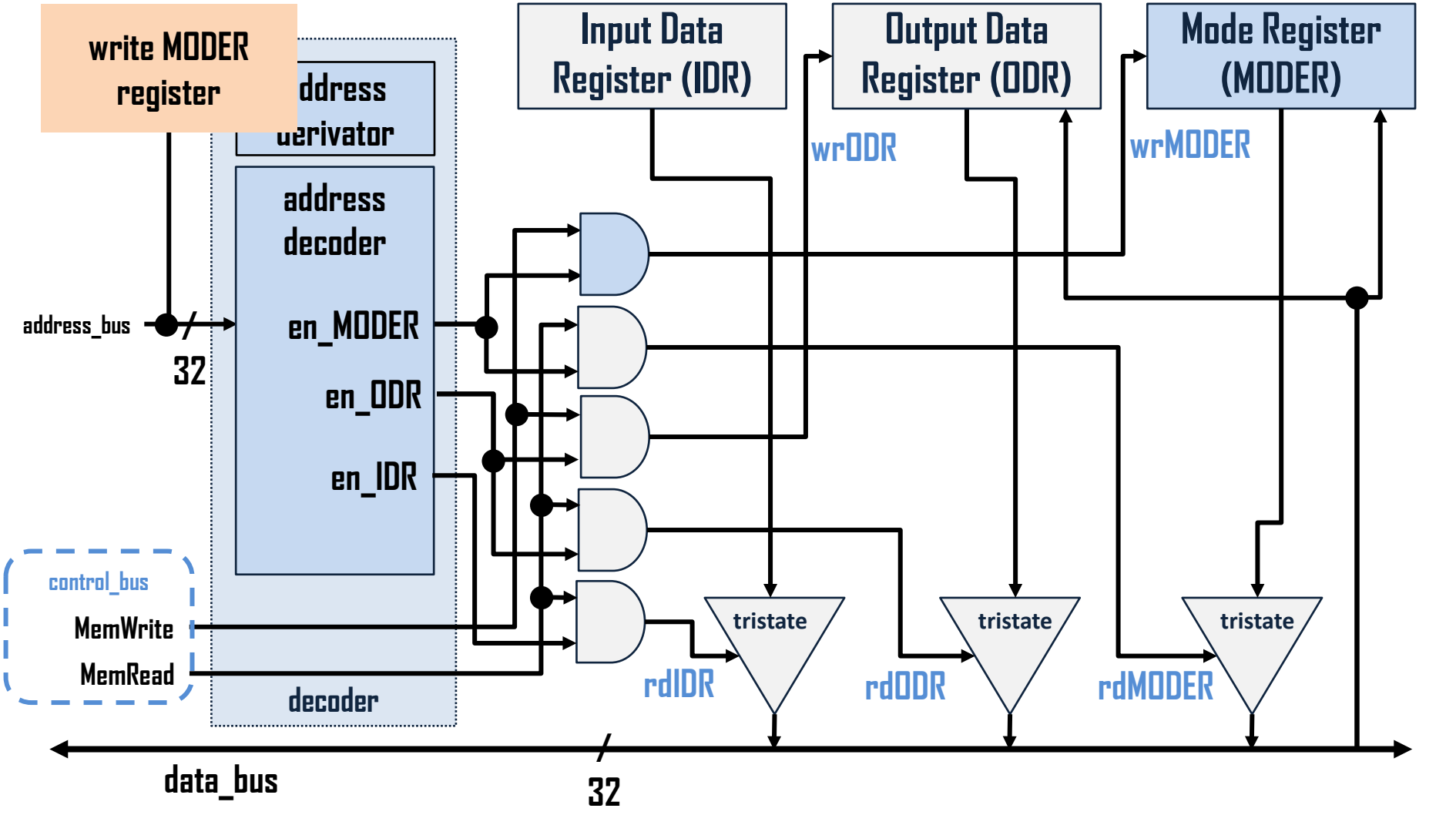
Mappaggio dell'I/O Parallelo



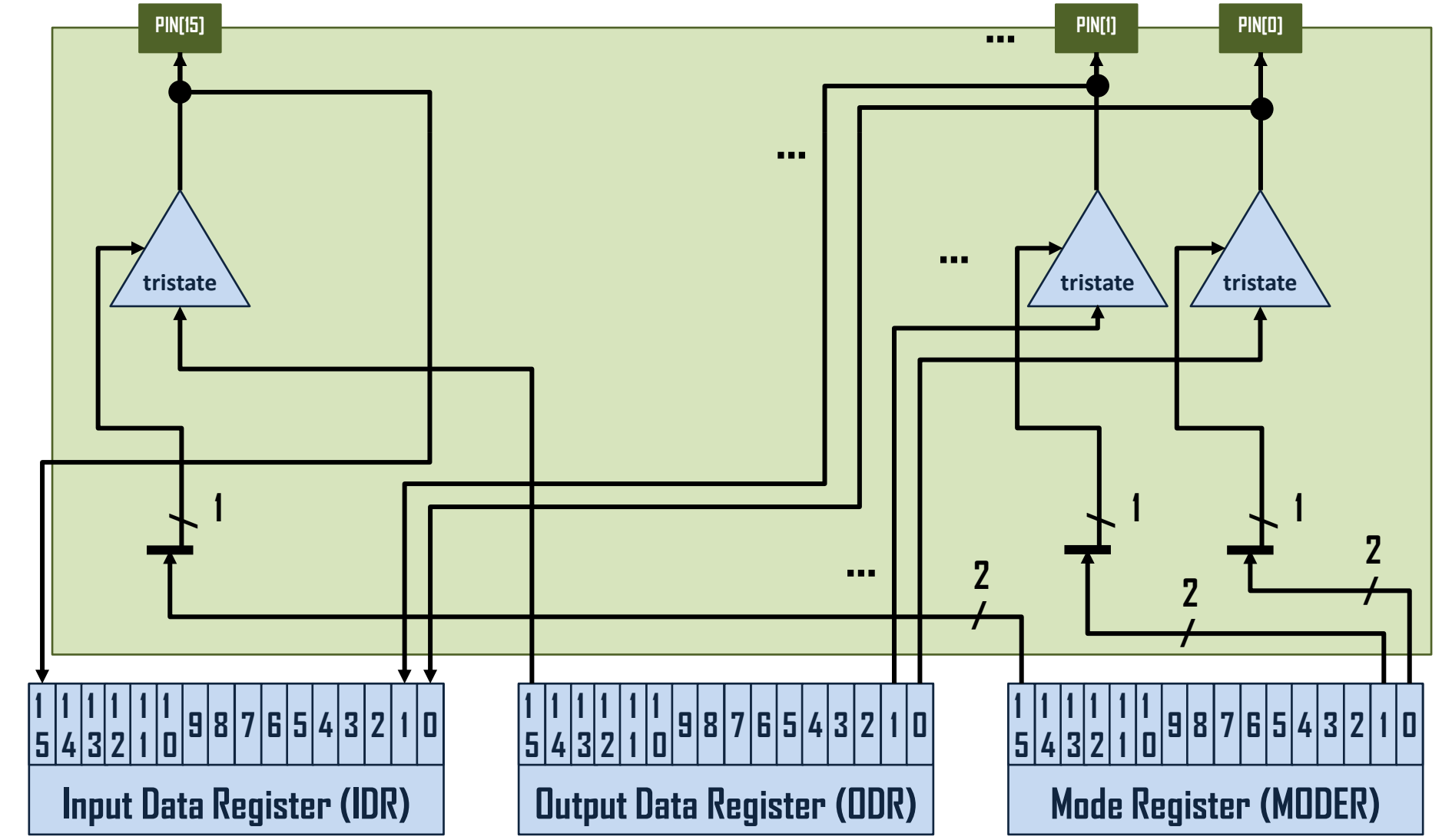
Mappaggio dell'I/O Parallelo



Mappaggio dell'I/O Parallelo



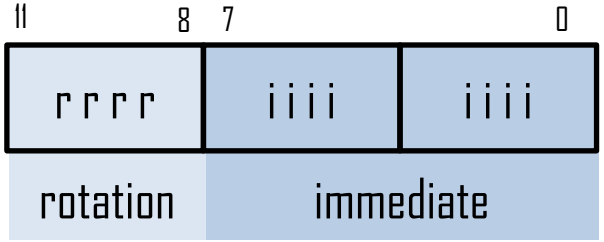
Mappaggio dell'I/O Parallelo



Indice

- Architettura con Bus e Mappaggio in Memoria
- Modifiche all'Architettura
  - Mappaggio della Memoria Dati
  - Mappaggio dell'I/O Parallelo
  - Rotazione dell'Immediate
- Esercitazione

Rotazione dell'Immediate



r \ K	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0																									i	i	i	i	i	i	i	i
1	i	i																									i	i	i	i	i	i
2	i	i	i	i																									i	i	i	i
3	i	i	i	i	i	i																									i	i
4	i	i	i	i	i	i	i	i																								
5			i	i	i	i	i	i	i	i																						
6					i	i	i	i	i	i	i	i																				
7							i	i	i	i	i	i	i	i																		
8									i	i	i	i	i	i	i	i																
9											i	i	i	i	i	i	i	i														
A													i	i	i	i	i	i	i	i												
B															i	i	i	i	i	i	i	i										
C																	i	i	i	i	i	i	i	i								
D																			i	i	i	i	i	i	i	i						
E																					i	i	i	i	i	i	i	i				
F																							i	i	i	i	i	i	i	i		

Moduli

- Moduli
  - decoder_dmem (dec_dmem)
  - decoder_port (dec_port_A)
  - port (port_A)
  - sign_extend*(sign_ext)
  - top* (dut)
  - tb_top* (top module testbench)

Indice

- Modifiche all'Architettura
  - Mappaggio della Memoria Dati
  - Mappaggio dell'I/O Parallelo
  - Rotazione dell'Immediate
- Esercitazione

Esercizio A

- Derivare il codice macchina ed eseguire la seguente funzione C nell'architettura semplificata di LEGv8 a disposizione, facendo attenzione al mappaggio della memoria dati

C

```
// X22 ← addr(A), A[0] ← 3, A[8] ← 10
// addr(A) ← dmem(0)
A[12] = A[0] + A[8];
```

boundary address	size	memory area/peripheral
0x20000000 – 0x20001FFF	8 kB	SRAM
0x48000000 – 0x480003FF	1 kB	GPIOA

Esercizio A

instr	Format 31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0	description
ADD	1 0 0 0	1 0 1 1	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn + Rm
SUB	1 1 0 0	1 0 1 1	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn – Rm
AND	1 0 0 0	1 0 1 0	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn & Rm
ORR	1 0 1 0	1 0 1 0	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn | Rm
LDUR	1 1 1 1	1 0 0 0	0 1 0 K	K K K K	K K K K	0 0 n n	n n n t	t t t t	Rt = dmem(Rn+K)
STUR	1 1 1 1	1 0 0 0	0 0 0 K	K K K K	K K K K	0 0 n n	n n n t	t t t t	dmem(Rn+K) = Rt
CBZ	1 0 1 1	0 1 0 0	K K K K	K K K K	K K K K	K K K K	K K K t	t t t t	if(Rt==0) then PC=PC+4+K*4 else PC=PC+4
B	0 0 0 1	0 1 K K	K K K K	K K K K	K K K K	K K K K	K K K K	K K K K	PC=PC+4+K*4
ADDI	1 0 0 1	0 0 0 1	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn + K
SUBI	1 1 0 1	0 0 0 1	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn – K
ANDI	1 0 0 1	0 0 1 0	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn & K
ORRI	1 0 1 1	0 0 1 0	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn | K

Esercizio A

C

```
// X22 ← addr(A), A[0] ← 3, A[8] ← 10
// addr(A) ← dmem(0)
A[12] = A[0] + A[8];
```

assembly

```
ADDI X22, XZR,#0x420; tmp0 ← addr(A)
LDUR X9, [X22,#32]  ; tmp1 ← A[8]
LDUR X10, [X22,#0]  ; tmp2 ← A[0]
ADD X9, X9, X10     ; tmp1 ← A[0] + A[8]
STUR X9, [X22,#48]  ; A[12] ← A[0] + A[8]
```

5 cicli

Esercizio A

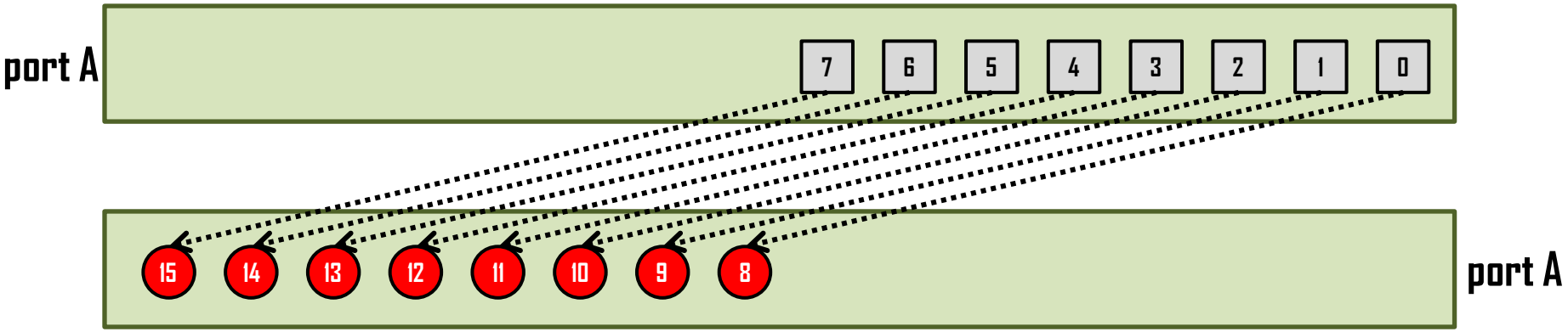
assembly	
ADDI X22, XZR,#0x420	; tmp0 ← addr(A)	911083F6
LDUR X9, [X22,#32]	; tmp1 ← A[8]	F84202C9
LDUR X10, [X22,#0]	; tmp2 ← A[0]	F84002CA
ADD X9, X9, X10	; tmp1 ← A[0] + A[8]	8B0A0129
STUR X9, [X22,#48]	; A[12] ← A[0] + A[8	F80302C9

Esercizio A

- imem.mem
  - @0x00 (0 in B, **0 in 32-bit words**): 911083F6
  - @0x04 (4 in B, **1 in 32-bit words**): F84202C9
  - @0x08 (8 in B, **2 in 32-bit words**): F84002CA
  - @0x0C (12 in B, **3 in 32-bit words**): 8B0A0129
  - @0x10 (16 in B, **4 in 32-bit words**): F80302C9
- dmem.mem
  - @0x00 (0 in B, **0 in 32-bit words**): 00000003
  - @0x20 (32 in B, **8 in 32-bit words**): 0000000A

Esercizio B

- Derivare un codice macchina in grado di leggere i valori dei primi 8 pin della porta A e scriverli nei restanti 8 pin della stessa porta per alimentare dei LED. Per farlo si modifichi l'architettura del LEGv8 semplificato per supportare le istruzioni di shift logico (LSL e LSR).



Esercizio B

instr	Format 31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0	description
ADD	1 0 0 0	1 0 1 1	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn + Rm
SUB	1 1 0 0	1 0 1 1	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn – Rm
AND	1 0 0 0	1 0 1 0	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn & Rm
ORR	1 0 1 0	1 0 1 0	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn | Rm
LSL	1 1 0 1	0 0 1 1	0 1 0 0	0 0 0 0	s s s s	s s n n	n n n d	d d d d	Rd = Rn << s
LSR	1 1 0 1	0 0 1 1	0 1 1 0	0 0 0 0	s s s s	s s n n	n n n d	d d d d	Rd = Rn >> s
LDUR	1 1 1 1	1 0 0 0	0 1 0 K	K K K K	K K K K	0 0 n n	n n n t	t t t t	Rt = dmem(Rn+K)
STUR	1 1 1 1	1 0 0 0	0 0 0 K	K K K K	K K K K	0 0 n n	n n n t	t t t t	dmem(Rn+K) = Rt
CBZ	1 0 1 1	0 1 0 0	K K K K	K K K K	K K K K	K K K K	K K K t	t t t t	if(Rt==0) then PC=PC+4+K*4 else PC=PC+4
B	0 0 0 1	0 1 K K	K K K K	K K K K	K K K K	K K K K	K K K K	K K K K	PC=PC+4+K*4
ADDI	1 0 0 1	0 0 0 1	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn + K
SUBI	1 1 0 1	0 0 0 1	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn – K
ANDI	1 0 0 1	0 0 1 0	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn & K
ORRI	1 0 1 1	0 0 1 0	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn | K

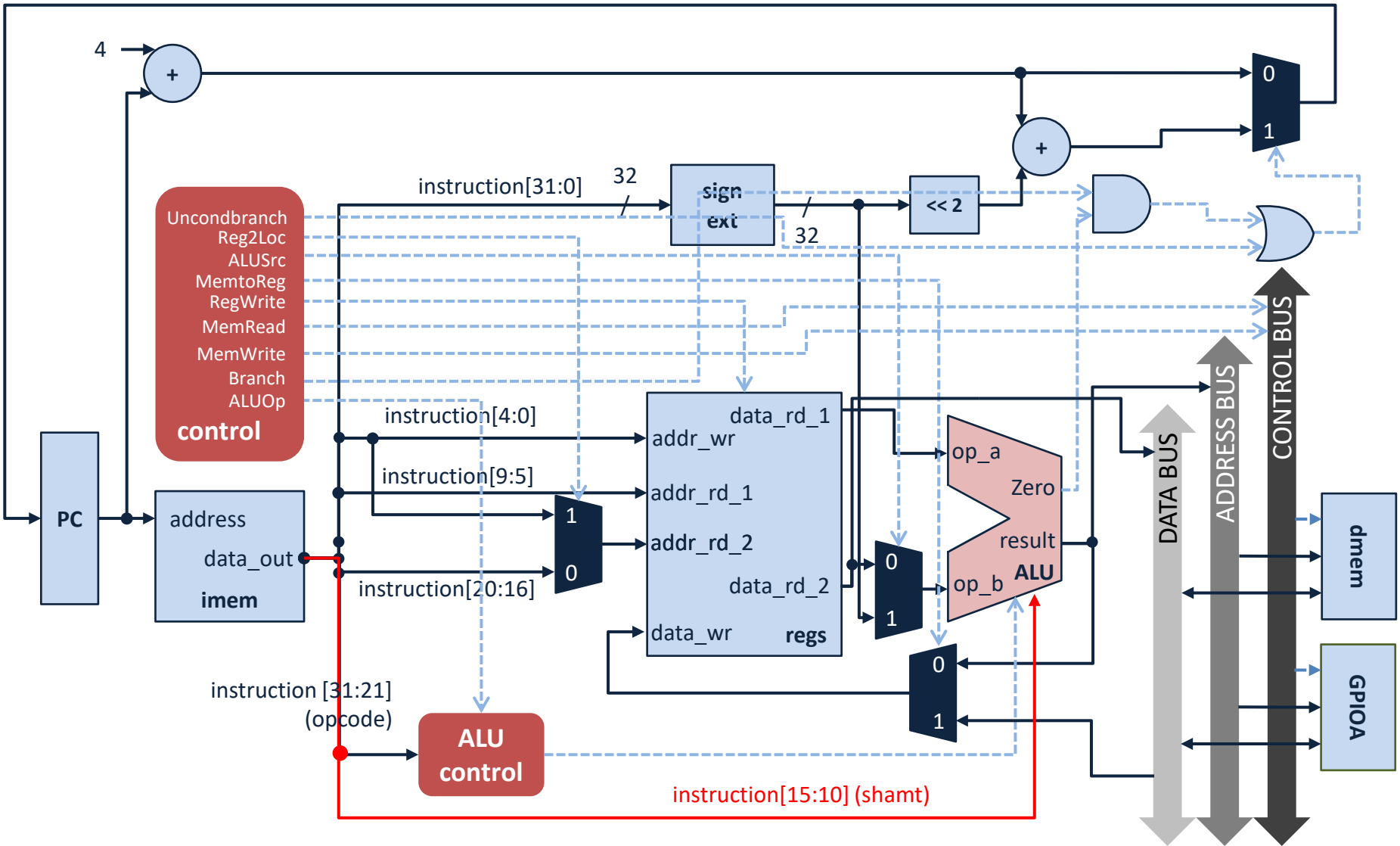
Esercizio B

boundary address	size	memory area/peripheral
0x20000000 – 0x20001FFF	8 kB	SRAM
0x48000000 – 0x480003FF	1 kB	GPIOA

offset	register	content			
0x00	GPIOx_MODER	MODER15[1:0]	...	MODER1[1:0]	MODER0[1:0]
0x10	GPIO_x_IDR			ID15:0	
0x14	GPIO_x_ODR			OD15:0	

MODERy[1:0]	I/O pin configuration
01	GP output
00	input

Esercizio B


4
+
+
0
1
instruction[31:0]
32
sign ext
32
<< 2
Uncondbranch
Reg2Loc
ALUSrc
MemtoReg
RegWrite
MemRead
MemWrite
Branch
ALUOp
control
instruction[4:0]
instruction[9:5]
instruction[20:16]
data_rd_1
addr_wr
addr_rd_1
addr_rd_2
data_rd_2
data_wr
regs
op_a
Zero
result
ALU
op_b
PC
address
data_out
imem
1
0
0
1
0
1
DATA BUS
ADDRESS BUS
CONTROL BUS
dmem
GPIOA
instruction [31:21]
(opcode)
ALU control
instruction[15:10] (shamt)

Esercizio B

ALU control			
instruction	ALUOp[1:0]	instruction[31:21]	ALU_opcode[3:0]
CBZ	01	XXXXXXXXXX	0010
LDUR/STUR	X1	XXXXXXXXXX	0111
ADD, ADDI	1X	10001011000 1001000100X	0010
SUB, SUBI	1X	11001011000 1101000100X	0110
AND, ANDI	1X	10001010000 1001001000X	0000
ORR, ORRI	1X	10101010000 1011001000X	0001
LSL	1X	11010011010	0100
LSR	1X	11010011011	0101

```
module alu #(...)(...
    input [5:0] shamt;
...);
...
always@(posedge clk)
     case(opcode)
        4'b0000 : result = op_a & op_b;
        4'b0001 : result = op_a | op_b;
        4'b0010 : result = op_a + op_b;
        4'b0100:    result = op_a << shamt;
        4'b0101:    result = op_a >> shamt;
        4'b0110 : result = op_a - op_b;
        4'b0111 : result = op_b;
       default:    result = {DATA_SIZE{1'bx}};
   endcase
...
endmodule
```

Esercizio B

```
module control (...);

...

always@(*)
    casex(instruction[31:20])
        // R-format
        11'b1xx0101x000:    ...
        // LDUR
        11'b11111000010:    ...
        // STUR
        11'b11111000000:    ...
        // CBZ
        11'b10110100xxx:    ...
        default:            ...
    endcase

...

endmodule
```

```
module control (...);

...

always@(*)
    casex(instruction[31:20])
        // R-format
        11'b1xxxx01x0xx:    ...
        // LDUR
        11'b11111000010:    ...
        // STUR
        11'b11111000000:    ...
        // CBZ
        11'b10110100xxx:    ...
        default:            ...
    endcase

...

endmodule
```

Esercizio B

assembly

```
ADDI X9, XZR, #0x448      ; tmp0 ← base addr A
ADDI X10, XZR, #0x455     ; tmp1 ← 0x55000000
ADDI X10, X10, #0x855     ; tmp1 ← 0x55550000
STUR X10, [X9,#0]         ; MODERA ← 0x55550000
LDUR X11, [X9,#0x10]      ; tmp2 ← port A value
LSL  X11, X11, #8         ; port A value << 8
STUR X11, [X9,#0x14]      ; port A value ← tmp2
```

8 cicli

Esercizio B

instr	Format 31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0	description
ADD	1 0 0 0	1 0 1 1	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn + Rm
SUB	1 1 0 0	1 0 1 1	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn – Rm
AND	1 0 0 0	1 0 1 0	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn & Rm
ORR	1 0 1 0	1 0 1 0	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn | Rm
LSL	1 1 0 1	0 0 1 1	0 1 0 0	0 0 0 0	s s s s	s s n n	n n n d	d d d d	Rd = Rn << s
LSR	1 1 0 1	0 0 1 1	0 1 1 0	0 0 0 0	s s s s	s s n n	n n n d	d d d d	Rd = Rn >> s
LDUR	1 1 1 1	1 0 0 0	0 1 0 K	K K K K	K K K K	0 0 n n	n n n t	t t t t	Rt = dmem(Rn+K)
STUR	1 1 1 1	1 0 0 0	0 0 0 K	K K K K	K K K K	0 0 n n	n n n t	t t t t	dmem(Rn+K) = Rt
CBZ	1 0 1 1	0 1 0 0	K K K K	K K K K	K K K K	K K K K	K K K t	t t t t	if(Rt==0) then PC=PC+4+K*4 else PC=PC+4
B	0 0 0 1	0 1 K K	K K K K	K K K K	K K K K	K K K K	K K K K	K K K K	PC=PC+4+K*4
ADDI	1 0 0 1	0 0 0 1	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn + K
SUBI	1 1 0 1	0 0 0 1	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn – K
ANDI	1 0 0 1	0 0 1 0	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn & K
ORRI	1 0 1 1	0 0 1 0	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn | K

Esercizio B

assembly

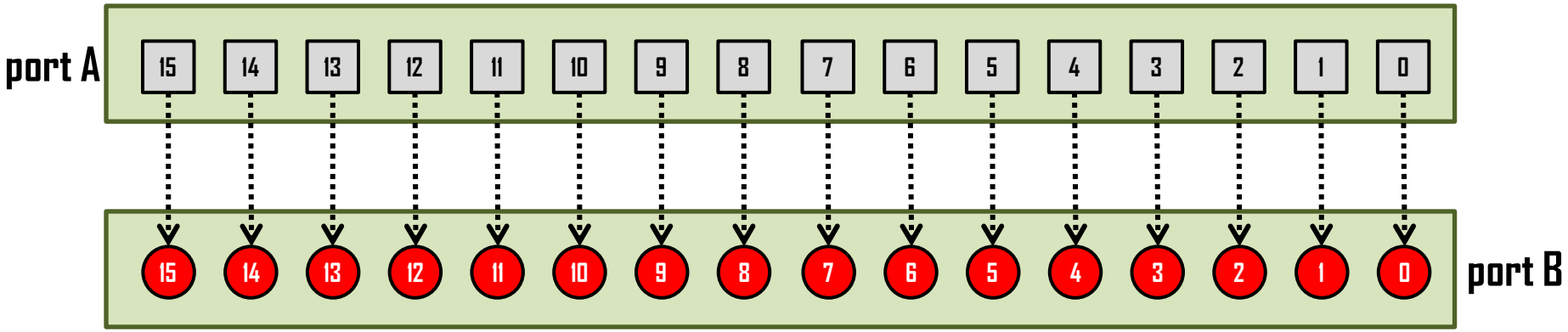
```
ADDI X9, XZR, #0x448      ; tmp0 ← base      91123E9
ADDI X10, XZR, #0x455     ; tmp1 ← 0x550     911157EA
ADDI X10, X10, #0x855     ; tmp1 ← 0x555     9121554A
STUR X10, [X9,#0]         ; MODERA ← 0x5     F800012A
LDUR X11, [X9,#0x10]      ; tmp2 ← port      F841012B
LSL  X11, X11, #8         ; tmp2 ← tmp2      D340216B
STUR X11, [X9,#0x14]      ; port A value     F801412B
```

Esercizio B

- imem.mem
  - @0x00 (0 in B, **0 in 32-bit words**): 911123E9
  - @0x04 (4 in B, **1 in 32-bit words**): 911157EA
  - @0x08 (8 in B, **2 in 32-bit words**): 9121554A
  - @0x0C (12 in B, **3 in 32-bit words**): F800012A
  - @0x10 (16 in B, **4 in 32-bit words**): F841012B
  - @0x14 (20 in B, **5 in 32-bit words**): D340216B
  - @0x18 (24 in B, **6 in 32-bit words**): F801412B
- dmem.mem

Esercizio C

- Derivare un codice macchina in grado di implementare il funzionamento mostrato di seguito nell'architettura semplificata di LEGv8 a disposizione (i valori letti dalla porta A vanno scritti nella porta B per alimentare dei LED). La porta B ed il relativo mappaggio vanno aggiunte all'architettura.



Esercizio C

instr	Format 31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0	description
ADD	1 0 0 0	1 0 1 1	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn + Rm
SUB	1 1 0 0	1 0 1 1	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn – Rm
AND	1 0 0 0	1 0 1 0	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn & Rm
ORR	1 0 1 0	1 0 1 0	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn | Rm
LDUR	1 1 1 1	1 0 0 0	0 1 0 K	K K K K	K K K K	0 0 n n	n n n t	t t t t	Rt = dmem(Rn+K)
STUR	1 1 1 1	1 0 0 0	0 0 0 K	K K K K	K K K K	0 0 n n	n n n t	t t t t	dmem(Rn+K) = Rt
CBZ	1 0 1 1	0 1 0 0	K K K K	K K K K	K K K K	K K K K	K K K t	t t t t	if(Rt==0) then PC=PC+4+K*4 else PC=PC+4
B	0 0 0 1	0 1 K K	K K K K	K K K K	K K K K	K K K K	K K K K	K K K K	PC=PC+4+K*4
ADDI	1 0 0 1	0 0 0 1	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn + K
SUBI	1 1 0 1	0 0 0 1	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn – K
ANDI	1 0 0 1	0 0 1 0	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn & K
ORRI	1 0 1 1	0 0 1 0	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn | K

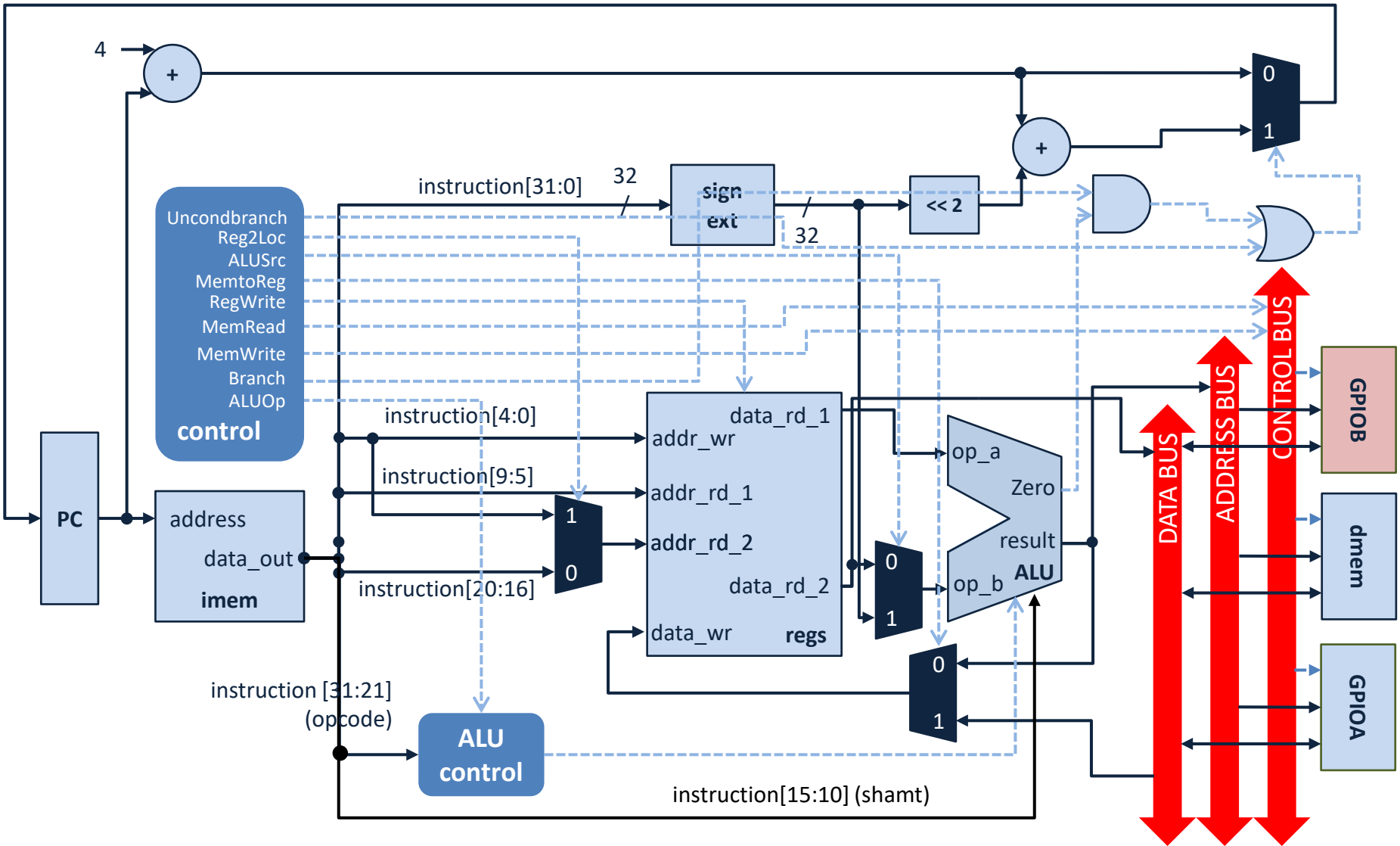
Esercizio C

boundary address	size	memory area/peripheral
0x20000000 – 0x20001FFF	8 kB	SRAM
0x48000000 – 0x480003FF	1 kB	GPIOA
0x48000400 – 0x480007FF	1 kB	GPIOB

offset	register	content			
0x00	GPIOx_MODER	MODER15[1:0]	...	MODER1[1:0]	MODER0[1:0]
0x10	GPIO_x_IDR			ID15:0	
0x14	GPIO_x_ODR			OD15:0	

MODERy[1:0]	I/O pin configuration
01	GP output
00	input

Esercizio C


4
+
instruction[31:0]
32
sign ext
32
<< 2
+
0
1
Uncondbranch
Reg2Loc
ALUSrc
MemtoReg
RegWrite
MemRead
MemWrite
Branch
ALUOp
control
PC
address
data_out
imem
instruction[4:0]
instruction[9:5]
instruction[20:16]
1
0
data_rd_1
addr_wr
addr_rd_1
addr_rd_2
data_rd_2
data_wr
regs
0
1
op_a
Zero
result
op_b
ALU
0
1
instruction [31:21]
(opcode)
ALU control
instruction[15:10] (shamt)
DATA BUS
ADDRESS BUS
CONTROL BUS
GPIOB
dmem
GPIOA

Esercizio C

assembly

```
ADDI X9, XZR, #0x448      ; tmp0 ← base addr A
ADDI X10, XZR, #0x455     ; tmp1 ← 0x55000000
ADDI X10, X10, #0x855     ; tmp1 ← 0x55550000
ADDI X10, X10, #0xC55     ; tmp1 ← 0x55555500
ADDI X10, X10, #0x055     ; tmp1 ← 0x55555555
ADDI X11, XZR, #0x448     ; tmp2 ← 0x48000000
ADDI X11, X11, #0xC04     ; tmp2 ← base addr B
STUR X10, [X11,#0x00]     ; MODERB ← 0x55555555
LDUR X12, [X9,#0x10]      ; tmp3 ← port A value
STUR X12, [X11,#0x14]     ; port B value ← tmp3
```

11 cicli

Esercizio C

instr	Format 31:28	27:24	23:20	19:16	15:12	11:8	7:4	3:0	description
ADD	1 0 0 0	1 0 1 1	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn + Rm
SUB	1 1 0 0	1 0 1 1	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn – Rm
AND	1 0 0 0	1 0 1 0	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn & Rm
ORR	1 0 1 0	1 0 1 0	0 0 0 m	m m m m	0 0 0 0	0 0 n n	n n n d	d d d d	Rd = Rn | Rm
LSL	1 1 0 1	0 0 1 1	0 1 0 0	0 0 0 0	s s s s	s s n n	n n n d	d d d d	Rd = Rn << s
LSR	1 1 0 1	0 0 1 1	0 1 1 0	0 0 0 0	s s s s	s s n n	n n n d	d d d d	Rd = Rn >> s
LDUR	1 1 1 1	1 0 0 0	0 1 0 K	K K K K	K K K K	0 0 n n	n n n t	t t t t	Rt = dmem(Rn+K)
STUR	1 1 1 1	1 0 0 0	0 0 0 K	K K K K	K K K K	0 0 n n	n n n t	t t t t	dmem(Rn+K) = Rt
CBZ	1 0 1 1	0 1 0 0	K K K K	K K K K	K K K K	K K K K	K K K t	t t t t	if(Rt==0) then PC=PC+4+K*4 else PC=PC+4
B	0 0 0 1	0 1 K K	K K K K	K K K K	K K K K	K K K K	K K K K	K K K K	PC=PC+4+K*4
ADDI	1 0 0 1	0 0 0 1	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn + K
SUBI	1 1 0 1	0 0 0 1	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn – K
ANDI	1 0 0 1	0 0 1 0	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn & K
ORRI	1 0 1 1	0 0 1 0	0 0 K K	K K K K	K K K K	K K n n	n n n d	d d d d	Rd = Rn | K

Esercizio C

assembly

```
ADDI X9, XZR, #0x448      ; tmp0 ← base    91123E9
ADDI X10, XZR, #0x455     ; tmp1 ← 0x550   911157EA
ADDI X10, X10, #0x855     ; tmp1 ← 0x555   9121554A
ADDI X10, X10, #0xC55     ; tmp1 ← 0x555   9131554A
ADDI X10, X10, #0x055     ; tmp1 ← 0x555   9101554A
ADDI X11, XZR, #0x448     ; tmp2 ← 0x480   91123EB
ADDI X11, X11, #0xE40     ; tmp2 ← base    9139016B
STUR X10, [X11,#0x00]     ; MODERB ← 0x5   F800016A
LDUR X12, [X9,#0x10]      ; tmp3 ← port    F841012C
STUR X12, [X11,#0x14]     ; port B value   F801416C
```

Esercizio C

- imem.mem
  - @0x00 (0 in B, **0 in 32-bit words**): 911123E9
  - @0x04 (4 in B, **1 in 32-bit words**): 911157EA
  - @0x08 (8 in B, **2 in 32-bit words**): 9121554A
  - @0x0C (12 in B, **3 in 32-bit words**): 9131554A
  - @0x10 (16 in B, **4 in 32-bit words**): 9101554A
  - @0x14 (20 in B, **5 in 32-bit words**): 911123EB
  - @0x18 (24 in B, **6 in 32-bit words**): 9139016B
  - @0x1C (28 in B, **7 in 32-bit words**): F800016A
  - @0x20 (32 in B, **8 in 32-bit words**): F841012C
  - @0x24 (36 in B, **9 in 32-bit words**): F801416C
- dmem.mem