Laboratory practice 6 (FINAL LAB)

MIPS multicycle

toc

MIPS Instruction set

Instruction Formats

R-Type:
arithmetic-logical

31 26	21	16	11	6	0
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-Type:
with memory
conditional branch
immediate

31 26	21	16	0
op	**rs**	**rt**	**addr/immediate**
6 bits	5 bits	5 bits	16 bits

J-Type:
Jump (unconditional)

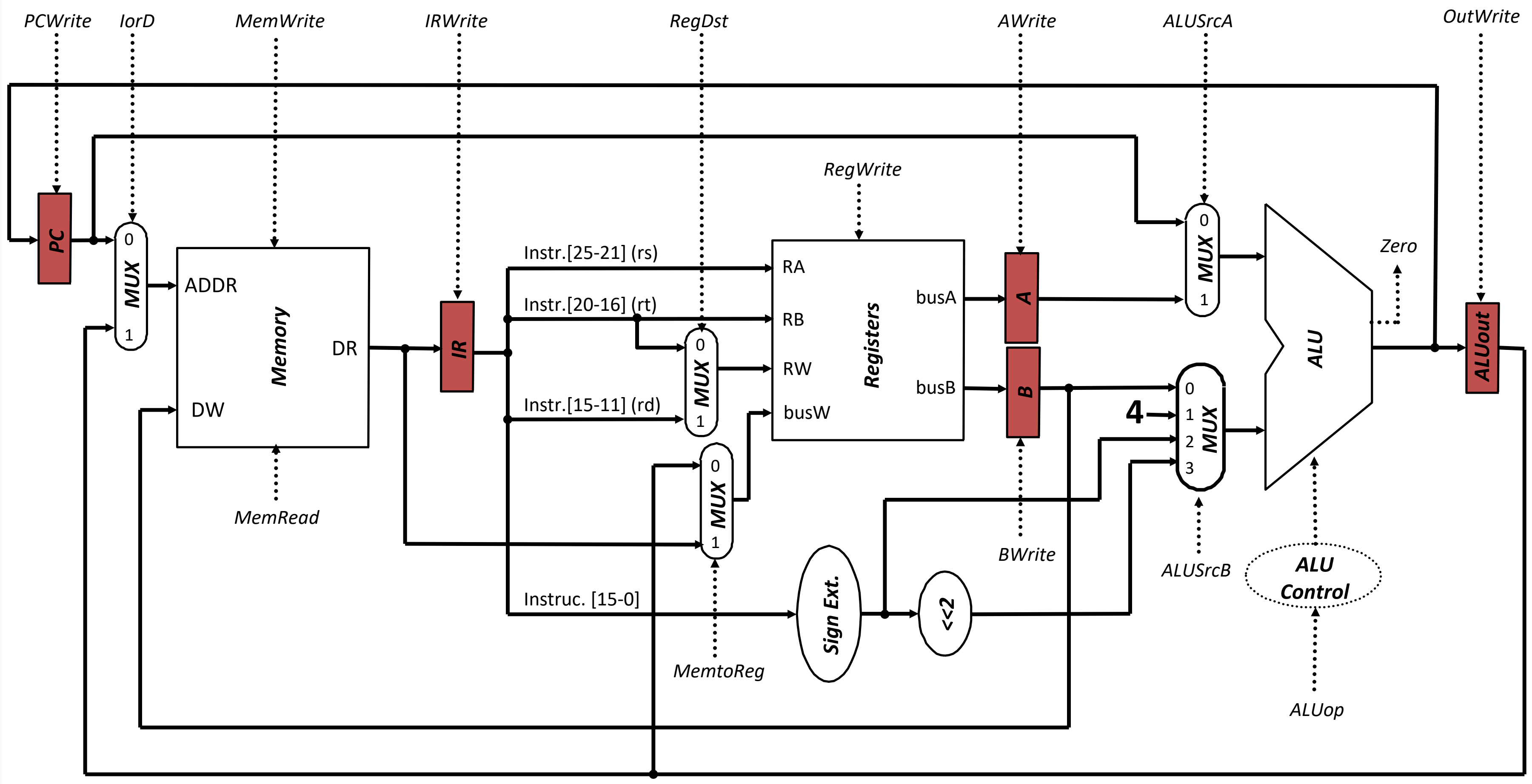
31 26	0
op	**target address**
6 bits	26 bits

The meaning of the fields is:

- **op**: opcode
- **rs**, **rt**, **rd**: first source register, second source registers and destination register
- **shamt**: shift amount indication (shift instructions)
- **funct**: arithmetic operation
- **immediate**: branch or address displacement
- **target address**: jump target address

toc

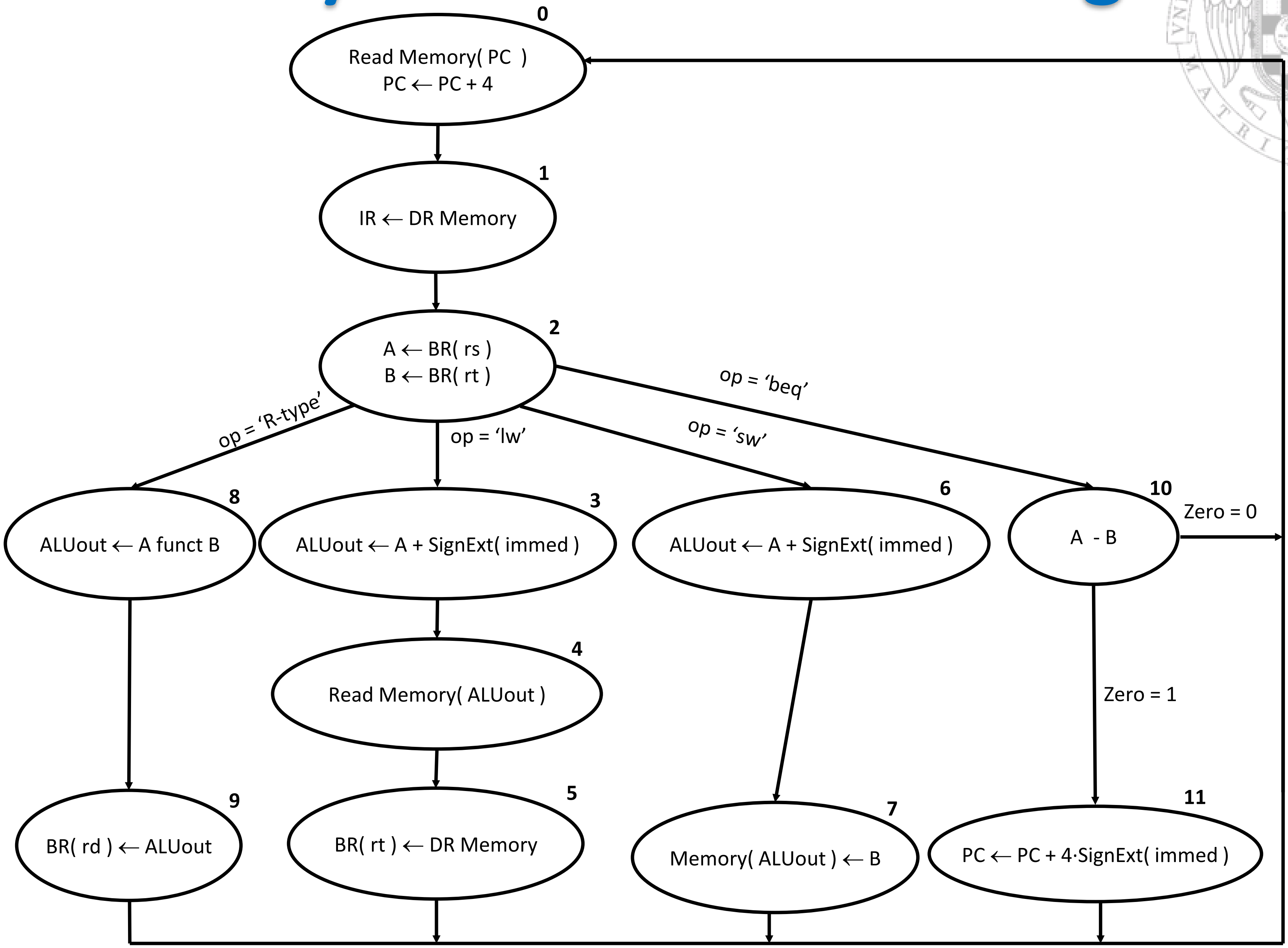
Multicycle Data Path design



Reduced MIPS Instruction set

- **Memory instructions (load & store)**:
  ```
  – lw rt, immed(rs): [rt<-Mem(rs+SignExt(immed)) ; PC<-PC+4]
  – sw rt, immed(rs): [Mem(rs+ SignExt(inmed))<-rt; PC<-PC+4]
  ```
- **Arithmetic-logical**:
  ```
  – add rd, rs, rt:   [rd <- rs  +  rt  ;  PC <- PC + 4]
  – sub rd, rs, rt:   [rd <- rs  -  rt  ;  PC <- PC + 4]
  – and rd, rs, rt:   [rd <- rs and rt  ;  PC <- PC + 4]
  – or  rd, rs, rt:   [rd <- rs  or rt  ;  PC <- PC + 4]
  – nor rd, rs, rt:   [rd <- rs nor rt  ;  PC <- PC + 4]
  – xor rd, rs, rt:   [rd <- rs xor rt  ;  PC <- PC + 4]
  ```
- **Jump (conditional):**
  ```
  – beq rs, rt, inmed:
     if( rs = rt ) then     PC <- PC + 4 + 4*SignExt(immed)
     else                   PC <- PC + 4
  ```

Multicycle Control Unit design



toc

Multicycle Control Unit design

Current State	op	Zero	Next State	IRWrite	PCWrite	AWrite	BWrite	ALUSrcA	ALUScrB	ALUOp	OutWrite	MemWrite	MemRead	IorD	MDRWrite	MemtoReg	RegDest	RegWrite
0000	XXXXXX	X	0001	0	1			0	01	00 (add)		0	1	0				0
0001	XXXXXX	X	0010	1	0							0	0	0				0
0010	100011 (lw)	X	0011	0	0	1	1					0	0					0
0010	101011 (sw)	X	0110															
0010	000000 (R-Type)	X	1000															
0010	000100 (beq)	X	1010															
0011	XXXXXX	X	0100	0	0			1	10	00 (add)	1	0	0					0
0100	XXXXXX	X	0101	0	0							0	1	1	1			0
0101	XXXXXX	X	0000	0	0							0	0			1	0	1
0110	XXXXXX	X	0111	0	0		0	1	10	00 (add)	1	0	0					0
0111	XXXXXX	X	0000	0	0							1	0	1				0
1000	XXXXXX	X	1001	0	0			1	00	10 (funct)	1	0	0					0
1001	XXXXXX	X	0000	0	0							0	0			0	1	1
1010	XXXXXX	0	0000	0	0			1	00	01 (sub)		0	0					0
1010	XXXXXX	1	1011															
1011	XXXXXX	X	0000	0	1			0	11	00 (add)		0	0					0

toc

Homework

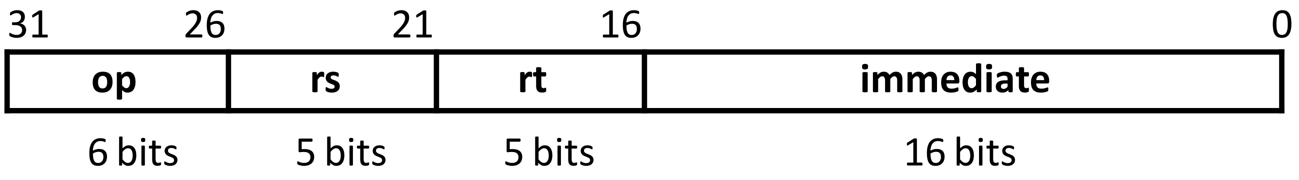
- Include tree new instructions:
  - **Move** with register
  - **Move** with immediate
  - **Jump** unconditional
- Test these instructions in the simulator:
  - Modify the `BlockRam` code to include the new instructions.
  - Configure and save wave configuration
    - Inspect RAM memory, PC, IR, register bank, etc.

toc

Move instructions

- **Move instruction with immediate:**
  - `mv rt, #immed : [rt <- SignExt(inmed); PC <- PC + 4]`

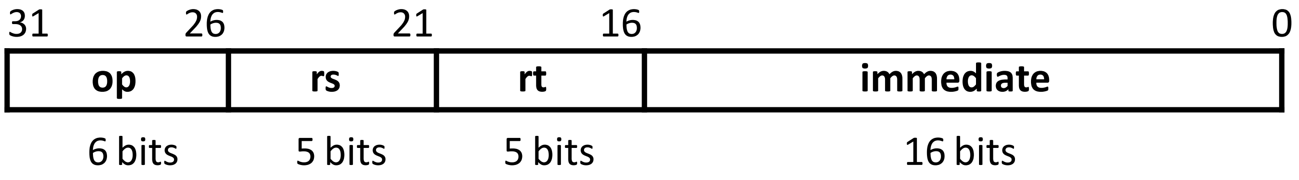
I-Type:
with memory
conditional branch
immediate



  - Opcode: `”010000”`

- **Move with register:**
  - `mv rt, rs :        [rt <- rs ; PC <- PC + 4]`

I-Type:
with memory
conditional branch
immediate



  - Opcode: `”010010”`

toc

Move instructions

- Modify the **`BlockRam`** program using the move instruction to set the registers to 0 and 1
  1. **`R3`** register will be initialized to 0 like this:
     - `mv r3, #0`

```
Opcode         rt     Immediate
010000 00000 00011 0000000000000000 -> x"40030000"
```

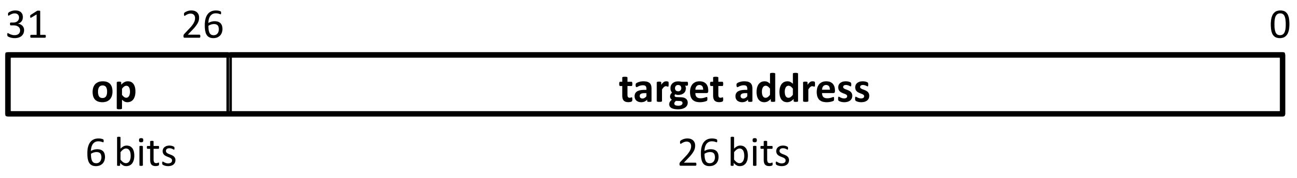
  2. Register **`R4`** will be initialized to 0 by copying the **`R3`** register like this:
     - `mv r4, r3`
  3. **`R2`** register will be initialized to 1 like this:
     - `mv r2, #1`

toc

Jump unconditional

- **Jump instruction :**
  - `j addr :          PC <- 0000 & addr & 00`

J-Type:
Jump (unconditional)



    - Opcode: “`000010`”

- Modify the **`BlockRam`** program using the jump instruction to replace:
  - `beq R0, R0, ADDR -> j ADDR`

toc

Grading

- Previous work is just to familiarize yourself with the processor.
  - Upload the vhdl files at the beginning of the session.
- During the exam you must implement two new instructions (1.25pts each)
- The laboratory exam is **NOT** recoverable

toc