

# Implémentation d'un MIPS32

---

## Introduction

Pirouz Bazargan-Sabet

# Effective Implementation of a 32-bit Processor

*Pirouz Bazargan Sabet*

University of Paris 6 - LIP6 - SoC



Pirouz Bazargan Sabet

Pirouz.Bazargan-Sabet@lip6.fr

March 2010

## Outline

- Introduction
- Architecture
- Implementation



Pirouz Bazargan Sabet

March 2010

## Introduction

### Architecture ?

- All aspects visible from the **USER's** (programmer) point of view
- External view
- Specifications of the processor

What the processor is supposed to do



Pirouz Bazargan Sabet

March 2010

## Introduction

### Implementation ?

- All aspects visible from the **DESIGNER's** point of view
- Internal view
- How much time does it take to perform some operation ?

Which hardware may be used and how it should be organized to make the specifications feasible



Pirouz Bazargan Sabet

March 2010

## Architecture

- ❑ Software visible registers
- ❑ Memory Addressing
- ❑ The instruction set
- ❑ The exception / reset mechanism



Pirouz Bazargan Sabet

March 2010

## Architecture

Architecture of the MIPS processor

Mips ?

- A 32-bit processor
- Defined in 1981 by the *Architecture Research Group* at the Stanford University (John Hennessy)



Pirouz Bazargan Sabet

March 2010

## Architecture

**Simplified Mips-32 architecture**

- No floating point operations
- No virtual memory management



Pirouz Bazargan Sabet

March 2010

## Architecture

- ❑ **Software visible registers**
- ❑ Memory Addressing
- ❑ The instruction set
- ❑ The exception / reset mechanism



Pirouz Bazargan Sabet

March 2010

## Architecture

### Software visible registers

Registers that can be manipulated (written or read) in the assembly language



Pirouz Bazargan Sabet

March 2010

## Software Visible Registers

- 32 common 32-bit registers  
Integer Registers

$R_0 \dots R_{31}$

Addressable from their number



Pirouz Bazargan Sabet

March 2010

## Software Visible Registers

- $R_0$  : The Trash Register

A value written into  $R_0$  is lost  
 $R_0$  contains always 0

- $R_{31}$  : The Link Register

When a subroutine is called, the  
return address is saved into  $R_{31}$



Pirouz Bazargan Sabet

March 2010

## Software Visible Registers

- Two 32-bit registers : HI and LO  
Used by multiply and divide instructions

Multiply      HI    32 most significant bits  
                 LO    32 least significant bits

Divide         LO    Result  
                 HI    Remainder



Pirouz Bazargan Sabet

March 2010

## Software Visible Registers

- 32-bit special registers : Coprocessor 0 registers required for the implementation an operating system

Status	Status Register
Cause	Cause Register (cause of exceptions)
Ebase	Exception Base Register
Epc	Exception Program Counter (return address in case of exception)
Eepc	Error Exception Program Counter
BadVAddr	Bad Address Register (illegal memory address)



Pirouz Bazargan Sabet

March 2010

## Architecture

- Software visible registers
- Memory Addressing
- The instruction set
- The exception / reset mechanism



Pirouz Bazargan Sabet

March 2010

## Memory Addressing

- 32-bit address  $\Rightarrow$  4 Gbytes of memory space
- Read / Write operations
- 3 types of data : Byte
  - Half-word (2 bytes)
  - Word (4 bytes)



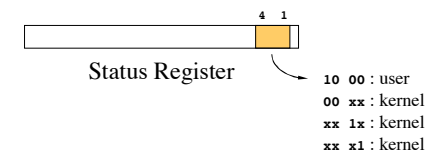
Pirouz Bazargan Sabet

March 2010

## Memory Addressing

- The processor can operate under 2 modes  
User / Kernel

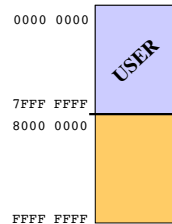
The current mode is defined by the Status Register



Pirouz Bazargan Sabet

March 2010

## Memory Addressing



The memory space is divided into 2 parts

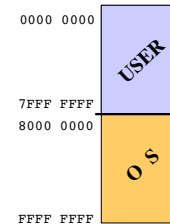
Under User mode the processor can only access the addresses ranging from 0000 0000 to 7FFF FFFF



Pirouz Bazargan Sabet

March 2010

## Memory Addressing



The OS protects the hardware against an error in a User program

The frontier protects the OS

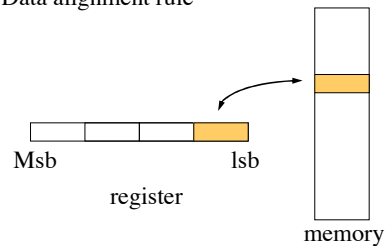


Pirouz Bazargan Sabet

March 2010

## Memory Addressing

### Data alignment rule

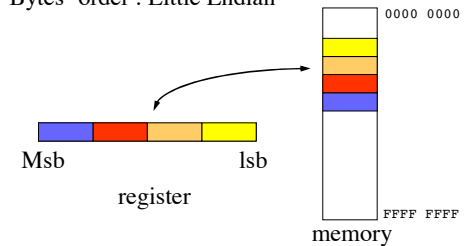


Pirouz Bazargan Sabet

March 2010

## Memory Addressing

### Bytes' order : Little Endian



Pirouz Bazargan Sabet

March 2010

## Memory Addressing

- Address alignment rule

The address of an object of **N** bytes must be multiple of **N**

Address of a Word      ⇒ multiple of 4  
Address of a Half-Word ⇒ multiple of 2  
Address of a Byte      ⇒ multiple of 1



Pirouz Bazargan Sabet

March 2010

## Architecture

- Software visible registers
- Memory Addressing
- The instruction set
- The exception / reset mechanism



Pirouz Bazargan Sabet

March 2010

## Instruction Set

### RISC Architecture

- Only simple instructions
- All the instructions have the same size (32 bits)
- 3-operand instructions (2 read, 1 write)
- No operation involving operands in memory - Only load and store operations



Pirouz Bazargan Sabet

March 2010

## Instruction Set

### 3 instruction formats

- R Register-register instructions
- I Immediate instructions
- J Jump instructions

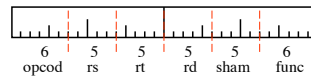


Pirouz Bazargan Sabet

March 2010

## Instruction Set

R format



opcode	Operation code
func	Function (opcode extension)
rs	# of source register
rt	# of source register
rd	# of destination register
sham	Shift amount (# of bit the opr. is shifted)

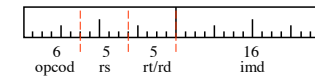


Pirouz Bazargan Sabet

March 2010

## Instruction Set

I format



opcode	Operation code
rs	# of source register
rt / rd	# of source or destination register
imd	Immediate value

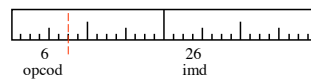


Pirouz Bazargan Sabet

March 2010

## Instruction Set

J format



opcode	Operation code
imd	Immediate value



Pirouz Bazargan Sabet

March 2010

## Instruction Set

- Arithmetic and logic instructions
- Memory access instructions
- Control instructions
- System instructions

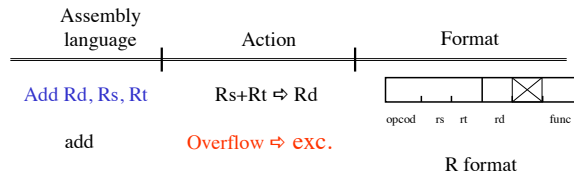


Pirouz Bazargan Sabet

March 2010



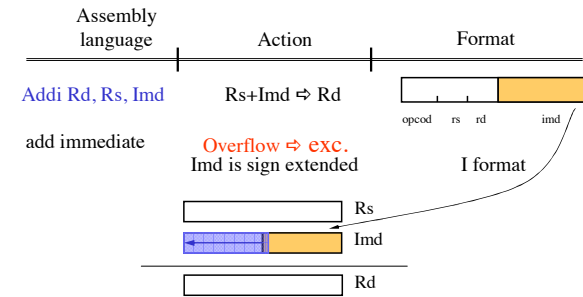
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

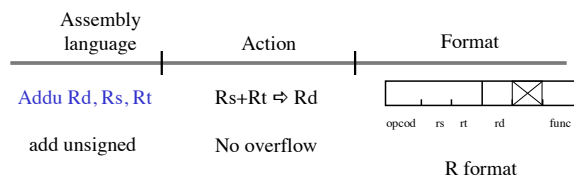
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

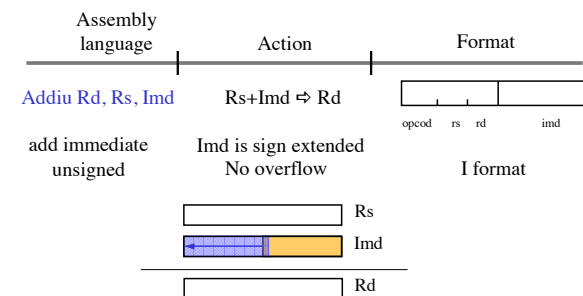
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

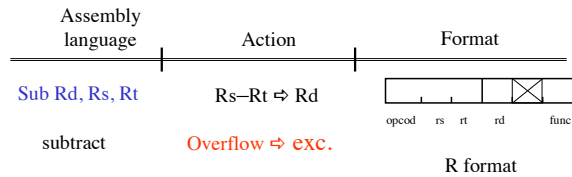
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

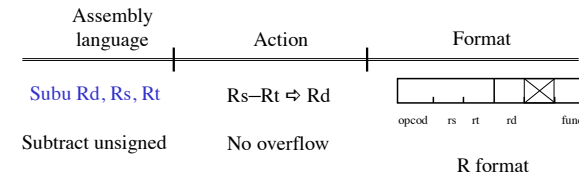
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

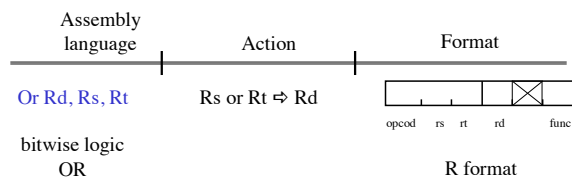
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

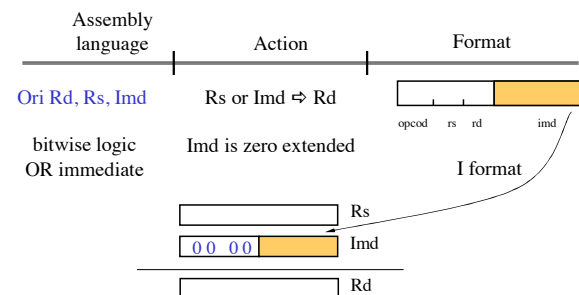
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

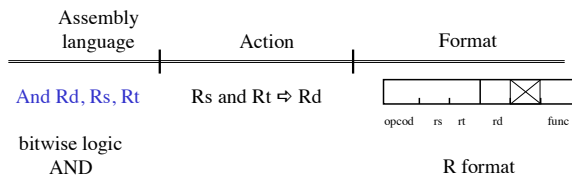
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

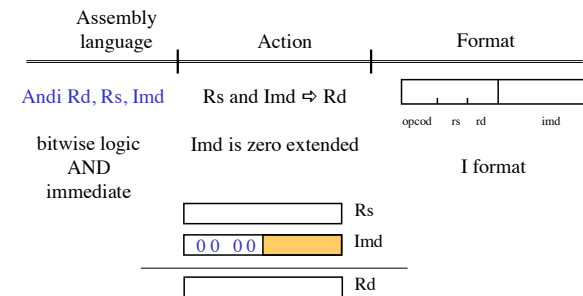
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

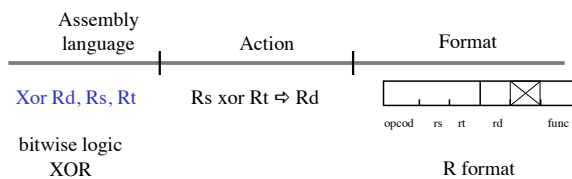
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

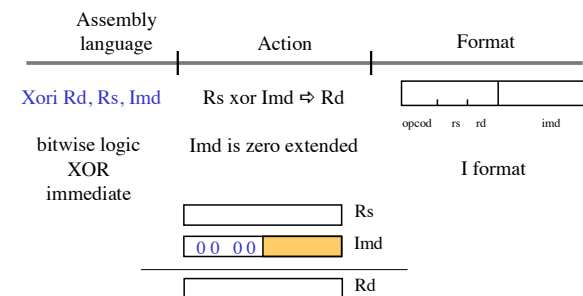
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

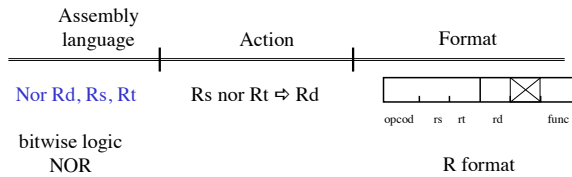
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

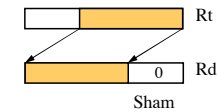
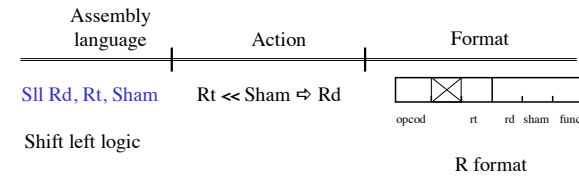
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

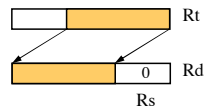
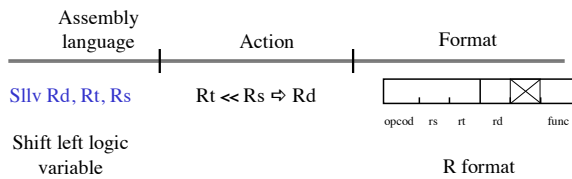
## Arithmetic and logic



Pirouz Bazargan Sabet

March 2010

## Arithmetic and logic



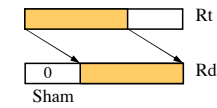
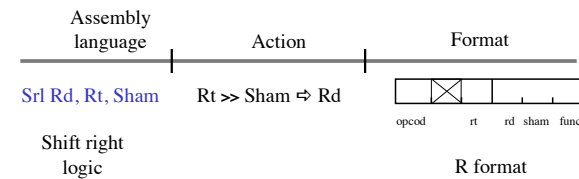
Only the 5 lsb of Rs are meaningful



Pirouz Bazargan Sabet

March 2010


## Arithmetic and logic

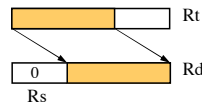


Pirouz Bazargan Sabet

March 2010

## Arithmetic and logic

Assembly language	Action	Format
<b>Srlv Rd, Rt, Rs</b>	$Rt \gg Rs \Rightarrow Rd$	
Shift right logic variable		R format



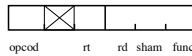
Only the 5 lsb of Rs are meaningful

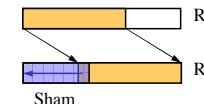


Pirouz Bazargan Sabet

March 2010

## Arithmetic and logic

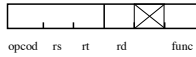
Assembly language	Action	Format
<b>Sra Rd, Rt, Sham</b>	$Rt \gg Sham \Rightarrow Rd$	
Shift right arithmetic		R format

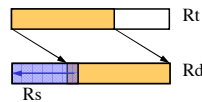


Pirouz Bazargan Sabet

March 2010

## Arithmetic and logic

Assembly language	Action	Format
<b>Srav Rd, Rt, Rs</b>	$Rt \gg Rs \Rightarrow Rd$	
Shift right arithmetic variable		R format



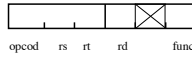
Only the 5 lsb of Rs are meaningful



Pirouz Bazargan Sabet

March 2010

## Arithmetic and logic

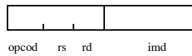
Assembly language	Action	Format
<b>Slt Rd, Rs, Rt</b>	$Rs < Rt ?$ Yes : $1 \Rightarrow Rd$ No : $0 \Rightarrow Rd$	
Set if less than	Operands are signed	R format



Pirouz Bazargan Sabet

March 2010

## Arithmetic and logic


Assembly language	Action	Format
<b>Slti Rd, Rs, Imd</b>	Rs < Imd ?	
Set if less than immediate	Yes : 1 ⇒ Rd No : 0 ⇒ Rd	I format
	Operands are signed Imd is sign extended	



Pirouz Bazargan Sabet

March 2010

## Arithmetic and logic

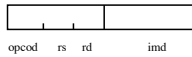
Assembly language	Action	Format
<b>Sltu Rd, Rs, Rt</b>	Rs < Rt ?	
Set if less than unsigned	Yes : 1 ⇒ Rd No : 0 ⇒ Rd	R format
	Operands are unsigned	



Pirouz Bazargan Sabet

March 2010

## Arithmetic and logic

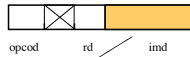
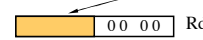
Assembly language	Action	Format
<b>Sltiu Rd, Rs, Imd</b>	Rs < Imd ?	
Set if less than immediate, unsigned	Yes : 1 ⇒ Rd No : 0 ⇒ Rd	I format
	Operands are unsigned Imd is sign extended	



Pirouz Bazargan Sabet

March 2010

## Arithmetic and logic

Assembly language	Action	Format
<b>Lui Rd, Imd</b>	Imd << 16 ⇒ Rd	
load upper immediate		I format
		



Pirouz Bazargan Sabet

March 2010

# Implémentation d'un MIPS32

---

## Accès mémoire

Pirouz Bazargan-Sabet

## Instruction Set

- Arithmetic and logic instructions
- Memory access instructions**
- Control instructions
- System instructions



Pirouz Bazargan Sabet

March 2010

## Instruction Set

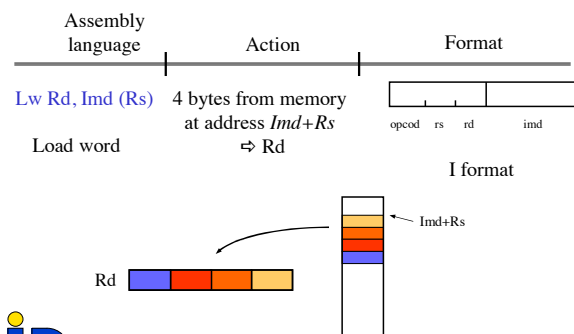
- Loads
- Stores



Pirouz Bazargan Sabet

March 2010

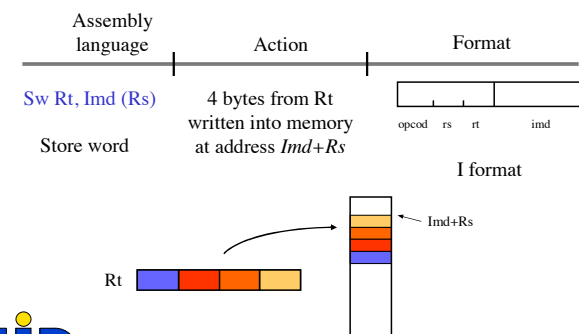
## Memory access



Pirouz Bazargan Sabet

March 2010

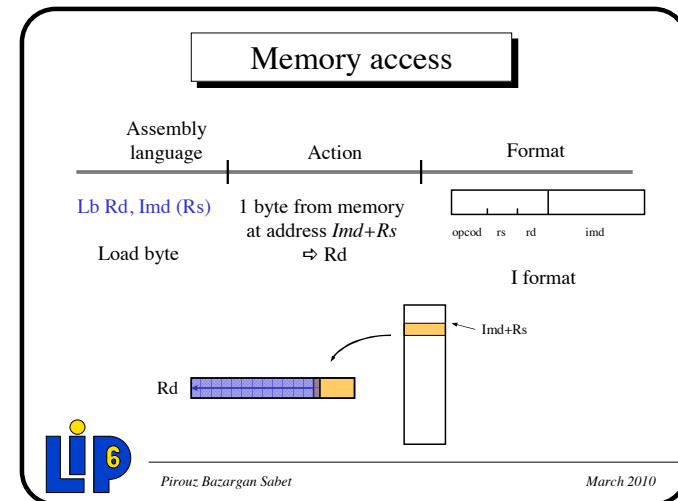
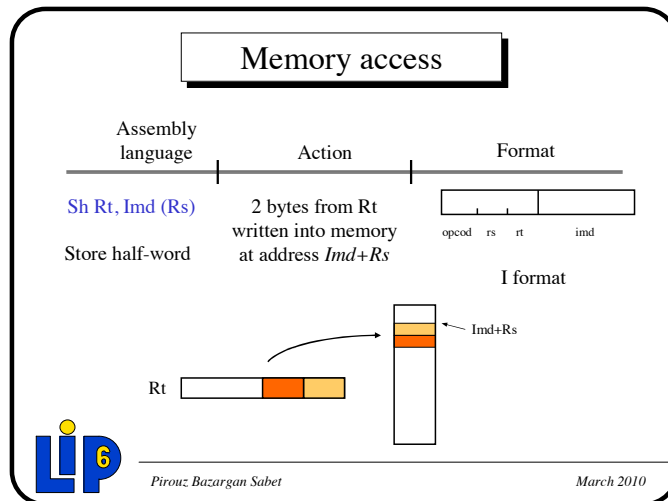
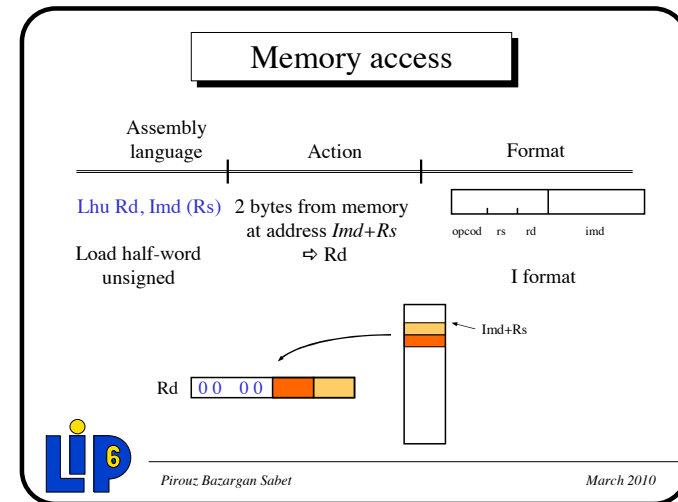
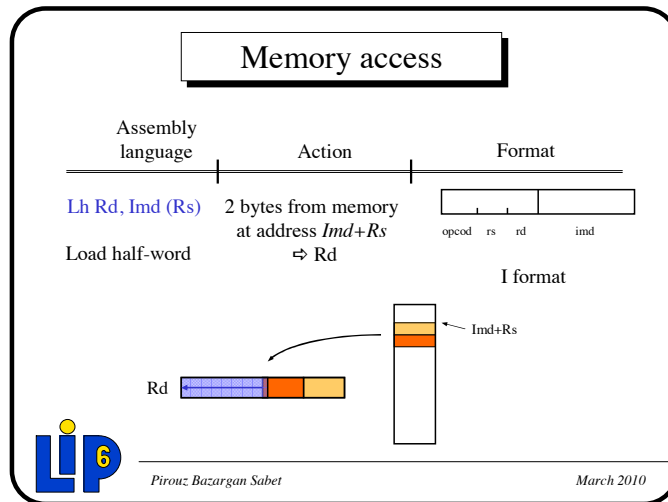
## Memory access

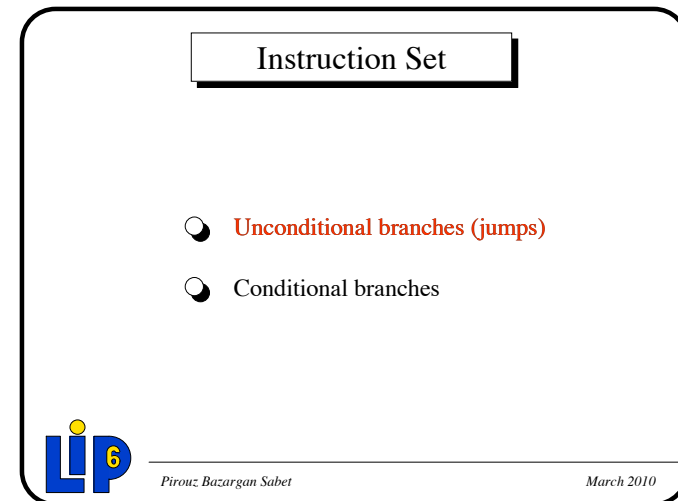
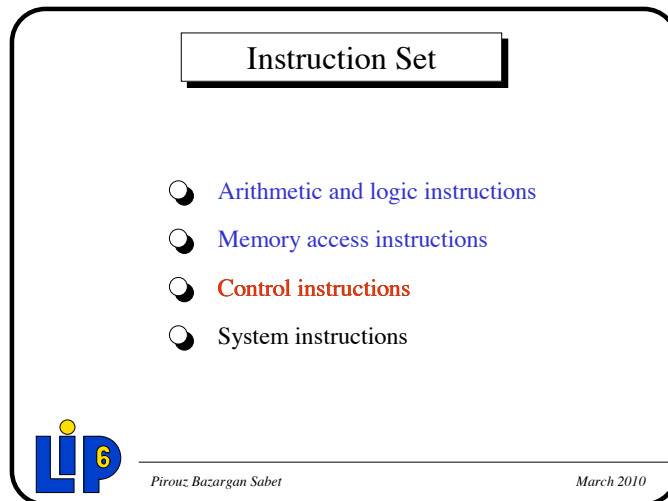
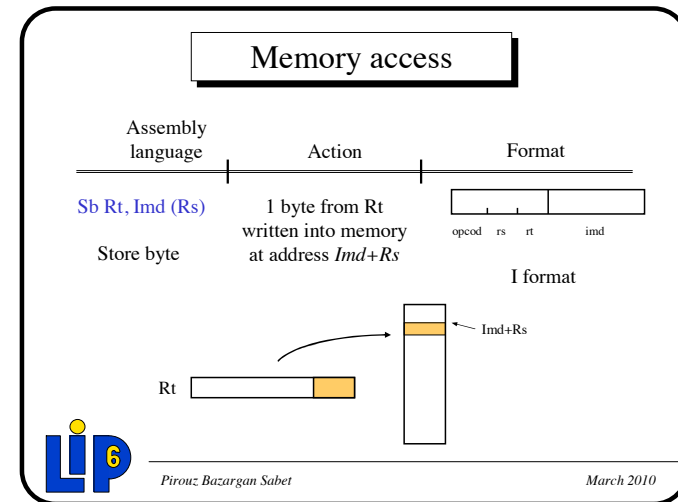
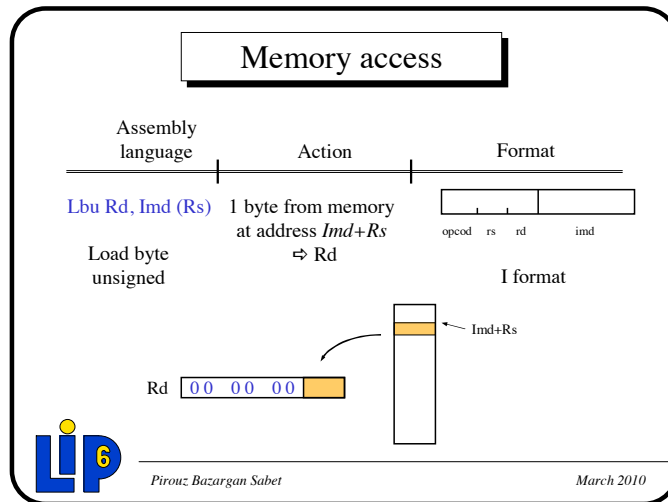


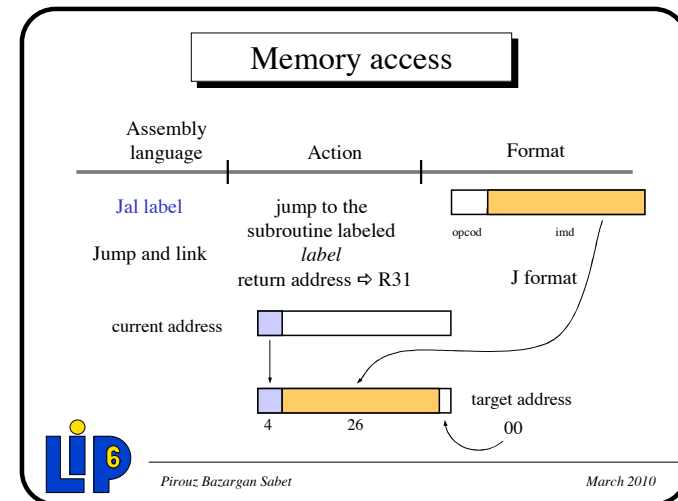
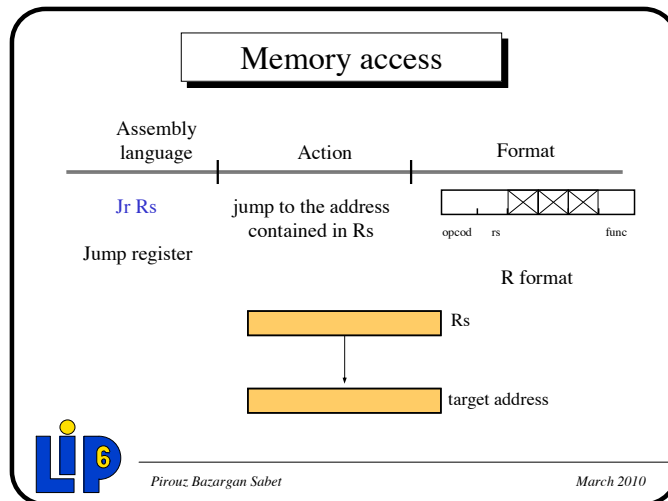
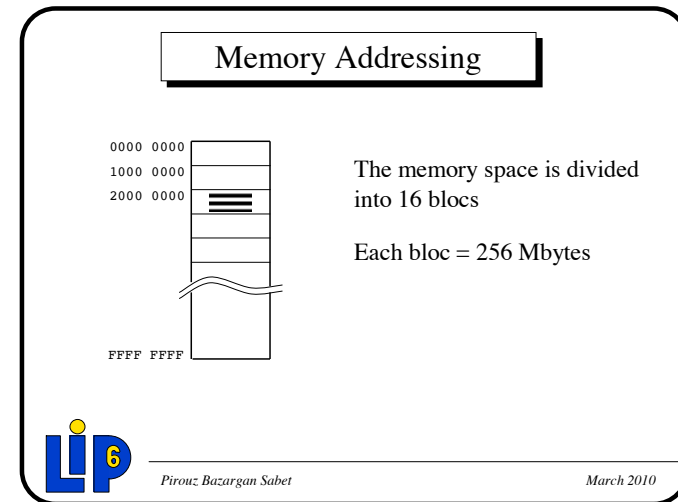
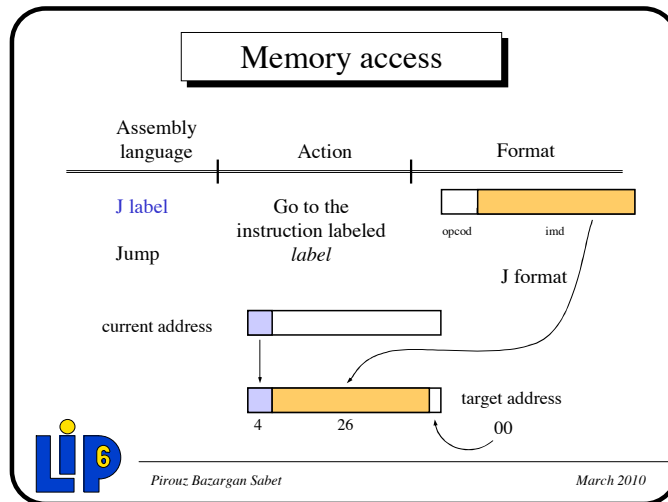
Pirouz Bazargan Sabet

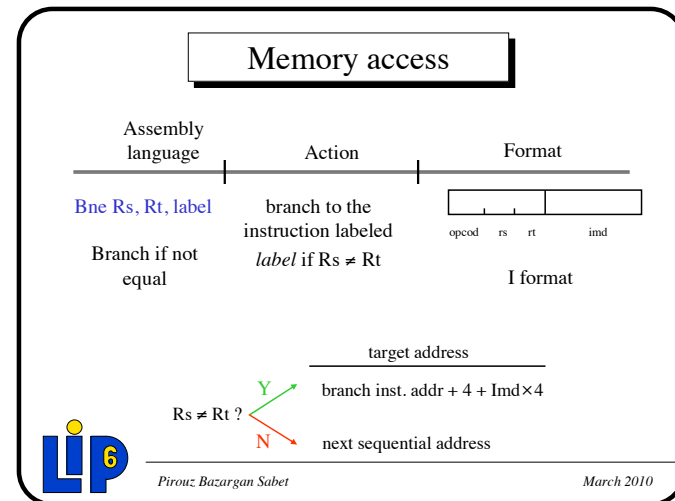
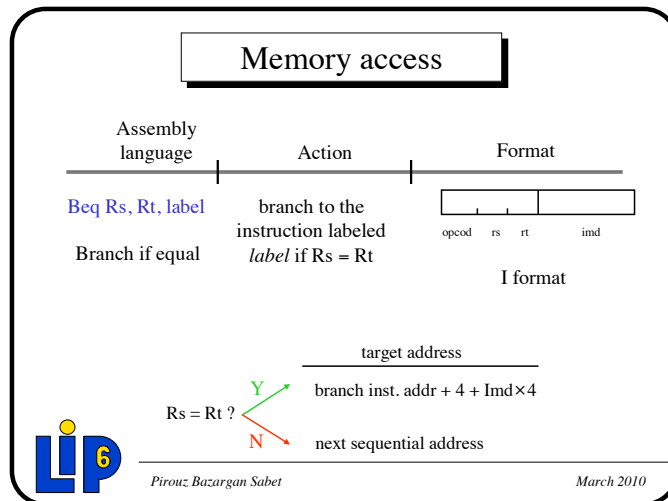
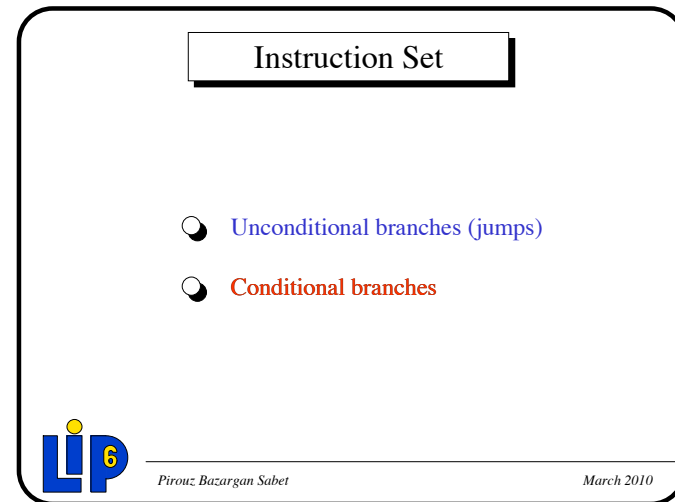
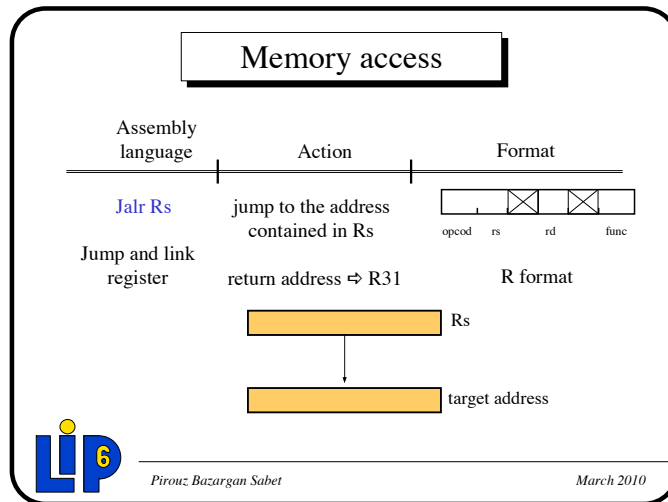
March 2010

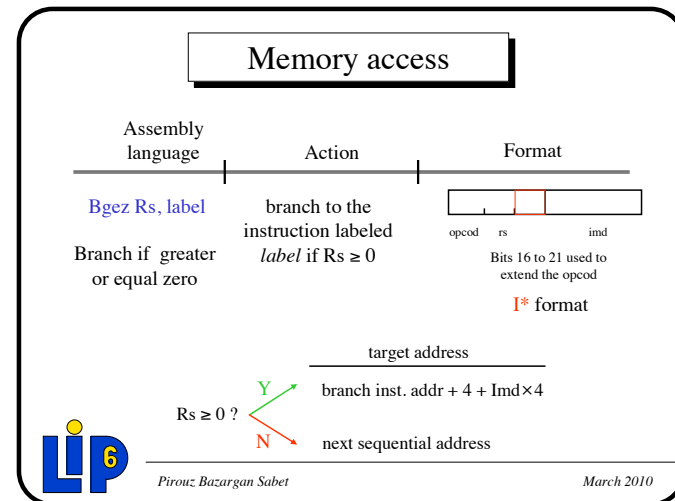
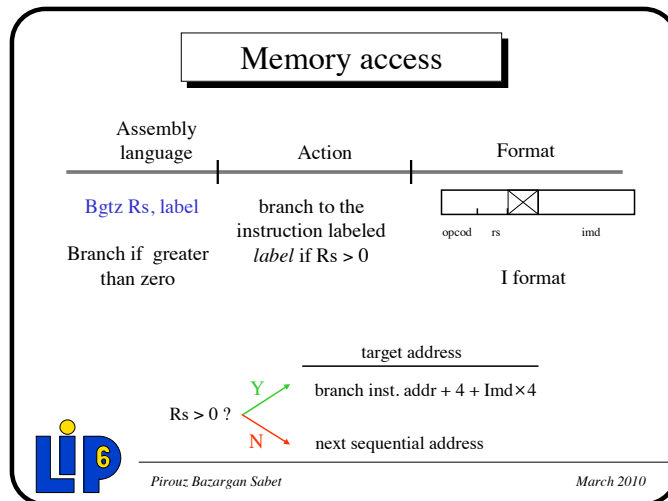
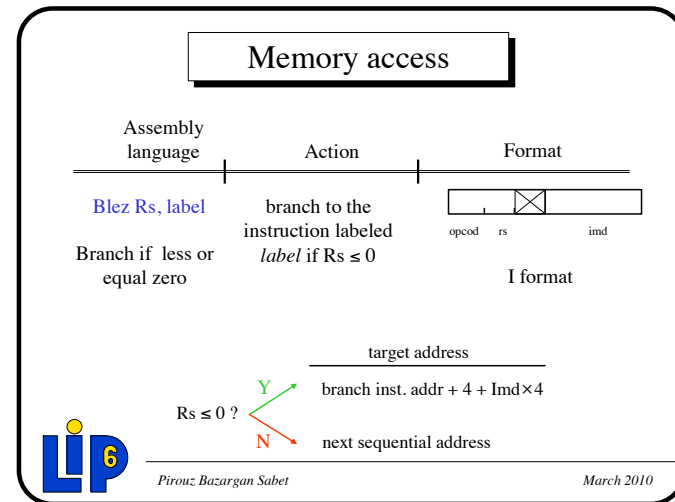
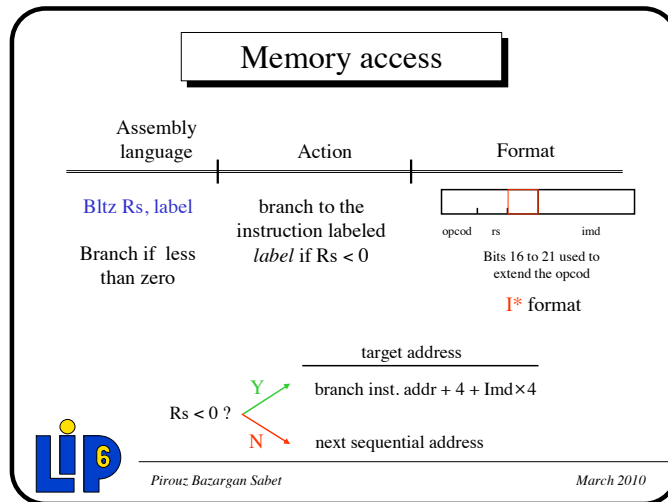












# Implémentation d'un MIPS32

---

## Exceptions et Reset

Pirouz Bazargan-Sabet

## Outline

- ❏ Introduction
- ❏ **Architecture**
- ❏ Implementation



Pirouz Bazargan Sabet

March 2010

## Architecture

- ❏ Software visible registers
- ❏ Memory Addressing
- ❏ The instruction set
- ❏ **The exception / reset mechanism**



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

Exception / Interrupt / Reset mechanism :

- **Reset mechanism**
- Interrupt mechanism
- Exception mechanism



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

### Reset

Re-initialize the system (the processor and all the other components of the system)

**Abort the execution of the current program**  
**All the data are lost**  
**Re-initialize the software (including the OS)**



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

**Reset** Abort the current program  
Jump to the Reset Handler

- Initialize the address of the next instruction : **0xBFC0 0000**
- Initialize the Status Register :

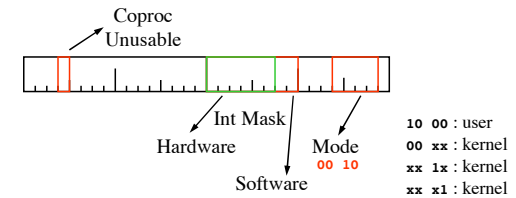


Pirouz Bazargan Sabet

March 2010

## Exception / Reset

### Status Register



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

**Reset** Abort the current program  
Jump to the Reset Handler

- Initialize the address of the next instruction : **0xBFC0 0000**
- Initialize the Status Register : **0x0000 0004**



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

Exception / Interrupt / Reset mechanism :

- Reset mechanism
- Interrupt mechanism
- Exception mechanism



Pirouz Bazargan Sabet

March 2010



## Exception / Reset

### Exception vs. Interrupt

Interrupts are events that require the processor to perform some operation

Interrupts are normal events during the life of a program

Exceptions are events that denote a malfunction in the program

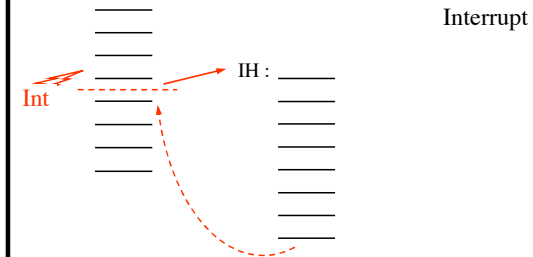
Exceptions are abnormal events during the life of a program



Pirouz Bazargan Sabet

March 2010

## Exception / Reset



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

**Interrupt**    Stop executing the current program  
                   Execute the Interrupt Handler  
                   Resume the interrupted program

- Initialize the address of the next instruction :  
**Ebase (31 downto 12) & X"180" : EXH-ADR**



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

**Reset**    Abort the current program  
                   Jump to the Reset Handler

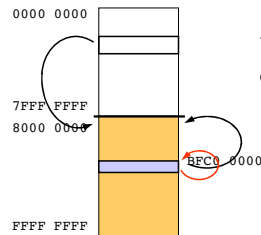
- Initialize the address of the next instruction : **0xBFC0 0000**
- Initialize the Status Register : **0x0000 0004**
- Initialize the Exception Base Register : **0x8000 0000**



Pirouz Bazargan Sabet

March 2010

## Exception / Reset



What happens if an interrupt occurs during the boot ?

- Go to **EXH-ADR**
- If the Os is not yet loaded ?
- Go to **0xBFC0 0380**

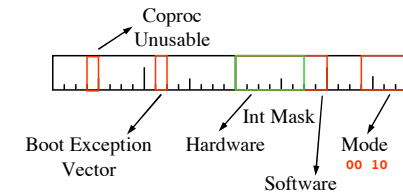


Pirouz Bazargan Sabet

March 2010

## Exception / Reset

### Status Register



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

**Reset** Abort the current program  
Jump to the Reset Handler

- Initialize the address of the next instruction : **0xBFC0 0000**
- Initialize the Status Register : **0x0040 0004**
- Initialize the Exception Base Register : **0x8000 0000**



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

**Interrupt** Stop executing the current program  
Execute the Interrupt Handler  
Resume the interrupted program

- Initialize the address of the next instruction :  
if *BootExcVect* = 0 **EXH-ADR**  
if *BootExcVect* = 1 **0xBFC0 0380**
- Set the SR : Mode (Kernel)

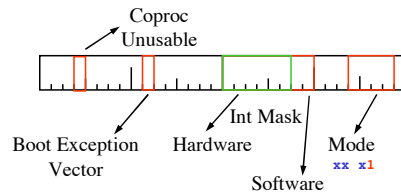


Pirouz Bazargan Sabet

March 2010

## Exception / Reset

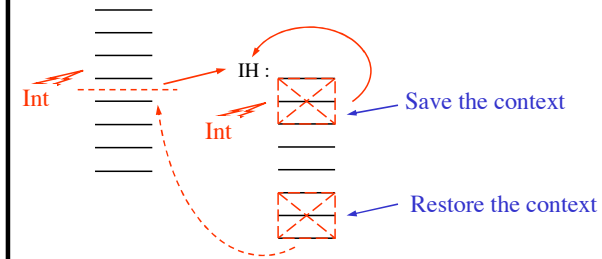
### Status Register



Pirouz Bazargan Sabet

March 2010

## Exception / Reset



Do not accept a new interrupt until the context is saved

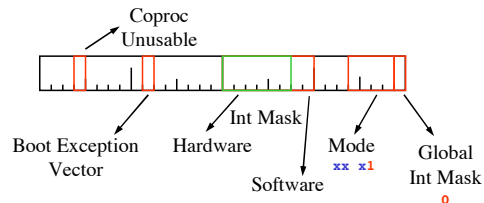


Pirouz Bazargan Sabet

March 2010

## Exception / Reset

### Status Register



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

**Interrupt** Stop executing the current program  
Execute the Interrupt Handler  
Resume the interrupted program

- Initialize the address of the next instruction :  
if *BootExcVect* = 0 **EXH-ADR**  
if *BootExcVect* = 1 **0xBFC0 0380**

- Set the SR : Mode (Kernel) , set the Global Int Mask



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

**Interrupt** Stop executing the current program  
Execute the Interrupt Handler  
Resume the interrupted program

- Initialize the address of the next instruction :  
if *BootExcVect* = 0 **EXH-ADR**  
if *BootExcVect* = 1 **0xBFC0 0380**
- Save the return address in EPC
- Set the SR : Mode (Kernel) , set the Global Int Mask
- Set the Cause

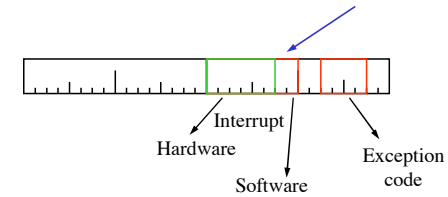


Pirouz Bazargan Sabet

March 2010

## Exception / Reset

- Cause Register Interrupt if set to 1



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

Exception causes

- Interrupt



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

**Reset** Abort the current program  
Jump to the Reset Handler

- Initialize the address of the next instruction : **0xBFC0 0000**
- Initialize the Status Register : **0x0040 0004**
- Initialize the Cause Register : **0x0000 0000**
- Save the return address into Eepc
- Initialize the Exception Base Register : **0x8000 0000**



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

Exception / Interrupt / Reset mechanism :

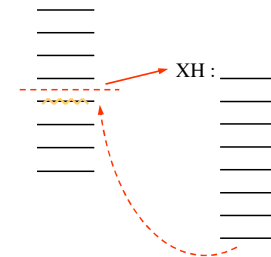
- Reset mechanism
- Interrupt mechanism
- Exception mechanism



Pirouz Bazargan Sabet

March 2010

## Exception / Reset



Exception

To preserve the integrity of the system, the faulty instruction should not be executed

In most of the cases a faulty program is not resumed, but killed



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

Exception   Stop executing the current program  
                   Execute the Exception Handler  
                   Resume the interrupted program

- Initialize the address of the next instruction :  
                   **EXH-ADR** or **0xBFC0 0380**
- Save the **faulty instruction's** address in EPC
- Set the SR : Mode (Kernel) , set the Global Int Mask
- Set the Cause
- Set the BadVAddr



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

Exception causes

- Interrupt
- Overflow
- Illegal read address (data or instruction)
- Illegal write address (data)
- Coprocessor unusable
- Unknown instruction
- Syscall
- Break
- Data bus error
- Instruction bus error



Pirouz Bazargan Sabet

March 2010

## Exception / Reset

### Exception vs. Interrupt

Interrupts are events that require the processor to perform some operation

- Interrupts are asynchronous
- ⇒ no emergency

Exceptions are events that denote a malfunction in the program

- An exception is an error
- ⇒ the faulty instruction should NOT be executed



Pirouz Bazargan Sabet

March 2010

# Implémentation d'un MIPS32

---

## Instructions système

Pirouz Bazargan-Sabet

## Instruction Set

- Arithmetic and logic instructions
- Memory access instructions
- Control instructions
- System instructions**



Pirouz Bazargan Sabet

March 2010

## System instructions

Assembly language	Action	Format
<b>Syscall</b>	Call a service of the operating system	R format
System Call	Stop executing the current program and jump to the Exception Handler	
	<b>Exception</b>	



Pirouz Bazargan Sabet

March 2010

## System instructions

Assembly language	Action	Format
<b>Break n</b>	Call the debugger through the operating system	R format
Break	Stop executing the current program and jump to the Exception Handler	
	<b>Exception</b>	



Pirouz Bazargan Sabet

March 2010

## System instructions

Assembly language	Action	Format
<b>Eret</b>	Return from exception	I* format
Exception return	Restore Status Register and jump to the interrupted program	
	<b>Exc if executed under User mode</b>	



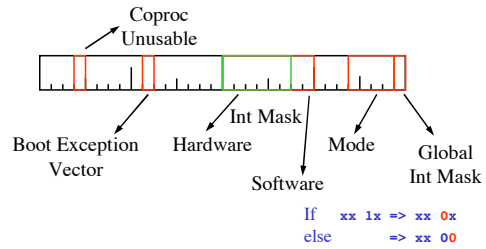
Pirouz Bazargan Sabet

March 2010



## System instructions

### Status Register



Pirouz Bazargan Sabet

March 2010

## System instructions

Assembly language	Action	Format
<b>Mfc0 Rd, S</b> Move from Coprocessor 0	Move a Copro 0 register into an integer register  Exc if executed under User mode	<p>Bits # 21 to 25 used to extend the opcode I* format</p>



Pirouz Bazargan Sabet

March 2010

## System instructions

Assembly language	Action	Format
<b>Mtc0 Rt, S</b> Move to Coprocessor 0	Move an integer register into a Copro 0 register  Exc if executed under User mode	<p>Bits # 21 to 25 used to extend the opcode I* format</p>



Pirouz Bazargan Sabet

March 2010

# Implémentation d'un MIPS32

---

## Implémentation

Pirouz Bazargan-Sabet

## Outline

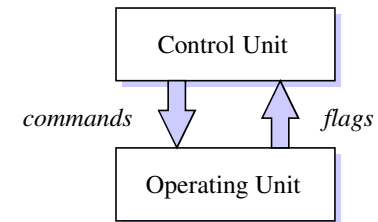
- Introduction
- Architecture
- Implementation



Pirouz Bazargan Sabet

March 2010

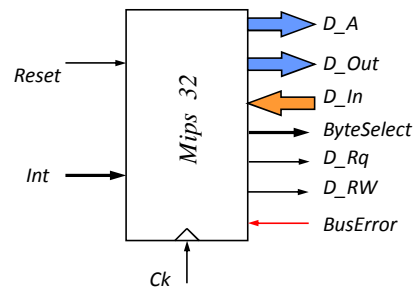
## Implementation



Pirouz Bazargan Sabet

March 2010

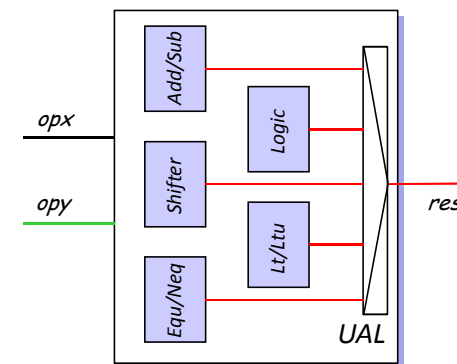
## Implementation



Pirouz Bazargan Sabet

March 2010

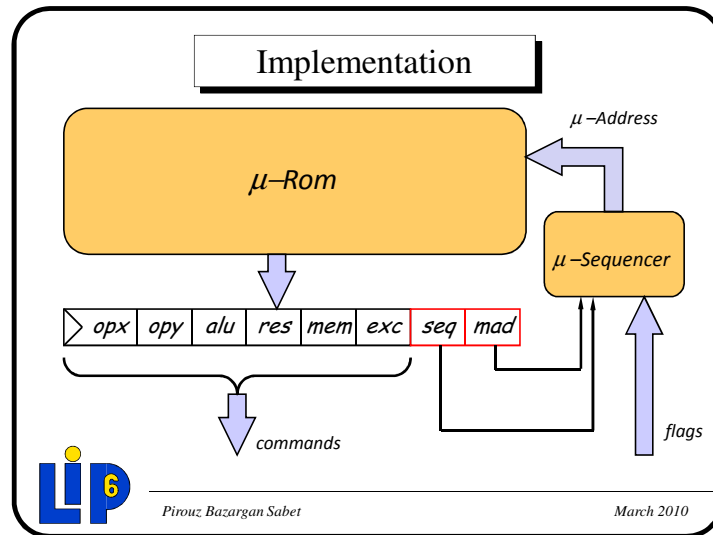
## Implementation



Pirouz Bazargan Sabet

March 2010





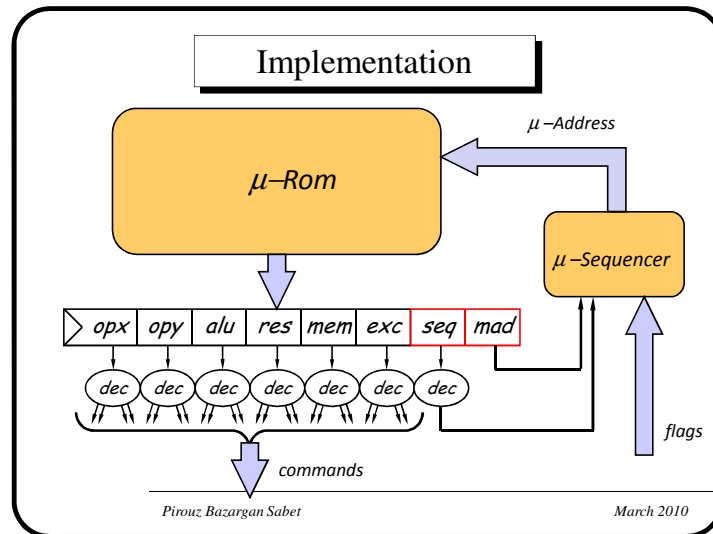
### Implementation

#### Opx μ-Functions

$x_{rs}$ : Reg (rs) → opx	$x_{ad}$ : Ad → opx
$x_{rt}$ : Reg (rt) → opx	$x_{ir}$ : I → opx
$x_{lo}$ : Lo → opx	$x_{din}$ : Din → opx
$x_{hi}$ : Hi → opx	$x_{c0}$ : 0 → opx
$x_{dad}$ : Dad → opx	$x_{c1}$ : 1 → opx
$x_{dot}$ : Dot → opx	$x_{c2}$ : 2 → opx
$x_{npc}$ : Nextpc → opx	$x_{c4}$ : 4 → opx
$x_{cpc}$ : Pc → opx	$x_{c16}$ : 16 → opx

**LIP** 6

Pirouz Bazargan Sabet March 2010



### Implementation

#### Opy μ-Functions

$y_{sr}$ : Status → opy	$y_{c0}$ : 0 → opy
$y_{cr}$ : Cause → opy	$y_{c1}$ : 1 → opy
$y_{ebs}$ : Ebase → opy	$y_{c2}$ : 2 → opy
$y_{bva}$ : BadVad → opy	$y_{c4}$ : 4 → opy
$y_{epc}$ : Epc → opy	$y_{c8}$ : 8 → opy
$y_{eep}$ : Eepc → opy	$y_{c16}$ : 16 → opy
$y_{dad}$ : Dad → opy	$y_{c24}$ : 24 → opy
$y_{dot}$ : Dot → opy	

**LIP** 6

Pirouz Bazargan Sabet March 2010

## Implementation

### Alu $\mu$ -Functions

<i>a_add</i> : $x + y \rightarrow res$	<i>a_sra</i> : $x \gg y \rightarrow res$
<i>a_sub</i> : $x - y \rightarrow res$	<i>a_eq</i> : $x = y ? \rightarrow res$
<i>a_and</i> : $x \text{ and } y \rightarrow res$	<i>a_ne</i> : $x \neq y ? \rightarrow res$
<i>a_or</i> : $x \text{ or } y \rightarrow res$	<i>a_lt</i> : $x < y ? \rightarrow res$
<i>a_xor</i> : $x \text{ xor } y \rightarrow res$	<i>a_ltu</i> : $x < y ? \rightarrow res$
<i>a_nor</i> : $x \text{ nor } y \rightarrow res$	
<i>a_sll</i> : $x \ll y \rightarrow res$	
<i>a_srl</i> : $x \gg y \rightarrow res$	



Pirouz Bazargan Sabet

March 2010

## Implementation

### Mem $\mu$ -Functions

*m\_nop* : *nop*  
*m\_rdi* : *read instruction*  
*m\_rdw* : *read word*  
*m\_wtb* : *write byte*  
*m\_wth* : *write half-word*  
*m\_wtw* : *write word*



Pirouz Bazargan Sabet

March 2010

## Implementation

### Res $\mu$ -Functions

<i>r_nop</i> : <i>nop</i>	<i>r_dot</i> : <i>res</i> $\rightarrow$ <i>Dot</i>
<i>r_rt</i> : <i>res</i> $\rightarrow$ <i>Reg (rt)</i>	<i>r_cr</i> : <i>res</i> $\rightarrow$ <i>Cause</i>
<i>r_rd</i> : <i>res</i> $\rightarrow$ <i>Reg (rd)</i>	<i>r_xcr</i> : <i>res</i> $\rightarrow$ <i>Cause</i>
<i>r_r31</i> : <i>res</i> $\rightarrow$ <i>Reg (31)</i>	<i>r_sr</i> : <i>res</i> $\rightarrow$ <i>Status</i>
<i>r_lo</i> : <i>res</i> $\rightarrow$ <i>Lo</i>	<i>r_ebs</i> : <i>res</i> $\rightarrow$ <i>Ebase</i>
<i>r_hi</i> : <i>res</i> $\rightarrow$ <i>Hi</i>	<i>r_bva</i> : <i>res</i> $\rightarrow$ <i>BadVad</i>
<i>r_dad</i> : <i>res</i> $\rightarrow$ <i>Dad</i>	<i>r_epc</i> : <i>res</i> $\rightarrow$ <i>Epc</i>
<i>r_npc</i> : <i>res</i> $\rightarrow$ <i>Nextpc</i>	<i>r_eep</i> : <i>res</i> $\rightarrow$ <i>Eepc</i>



Pirouz Bazargan Sabet

March 2010

## Implementation

### Exc $\mu$ -Functions

<i>e_nop</i> : <i>nop</i>	<i>e_isw</i> : <i>illegal store-word addr</i>
<i>e_ovr</i> : <i>overflow</i>	<i>e_ish</i> : <i>illegal store-half addr</i>
<i>e_sys</i> : <i>syscall</i>	<i>e_isb</i> : <i>illegal store-byte addr</i>
<i>e_brk</i> : <i>break</i>	<i>e_unk</i> : <i>unknown instruction</i>
<i>e_ia</i> : <i>illegal instr addr</i>	<i>e_cop</i> : <i>coprocessor unusable</i>
<i>e_ilw</i> : <i>illegal load-word addr</i>	<i>e_ibe</i> : <i>instr bus error</i>
<i>e_ilh</i> : <i>illegal load-half addr</i>	<i>e_dbe</i> : <i>data bus error</i>
<i>e_ilb</i> : <i>illegal load-byte addr</i>	<i>e_clr</i> : <i>clear exceptions</i>



Pirouz Bazargan Sabet

March 2010

## Implementation

### *Seq $\mu$ -Functions*

<i>s_opc</i>	: test opcode	...	xxx	xxx	...
<i>s_fun</i>	: test func	...	xxx	xxx	...
<i>s_jmp</i>	: jump	...	...	...	...
<i>s_exc</i>	: test exception	...	...	...	..x
<i>s_eqz</i>	: test previous res = 0	...	...	...	..x

. Comes from MAD field

x Comes from specified condition



Pirouz Bazargan Sabet

March 2010

# Implémentation d'un MIPS32

---

## Micro-programmation

Pirouz Bazargan-Sabet



## Outline

- Introduction
- Architecture
- **Implementation**



Pirouz Bazargan Sabet

March 2010

## Micro – programming

*Addu rd, rs, rt*

OPX	OPY	ALU	RES	MEM	EXC	SEQ
Rt	C0	Add	Dad	Nop	Nop	Jmp
Rs	Dad	Add	Rd	Nop	Nop	Jmp
Npc	C4	Add	Npc	Nop	Nop	Jmp
C0	C0	Add	Nop	Rdi	Nop	Jmp



Pirouz Bazargan Sabet

March 2010

## Micro – programming

*Addu rd, rs, rt*

OPX	OPY	ALU	RES	MEM	EXC	SEQ
Rt	C0	Add	Dad	Nop	Nop	Jmp
Rs	Dad	Add	Rd	Nop	Nop	Jmp

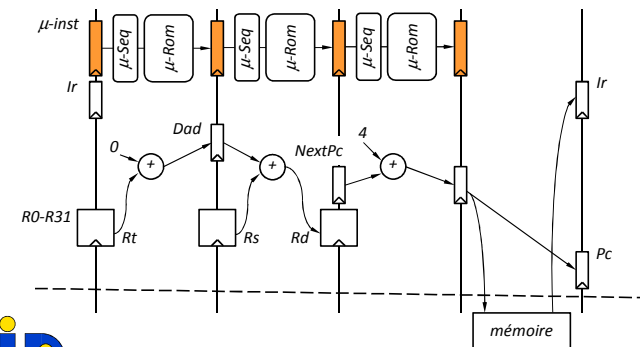


Pirouz Bazargan Sabet

March 2010

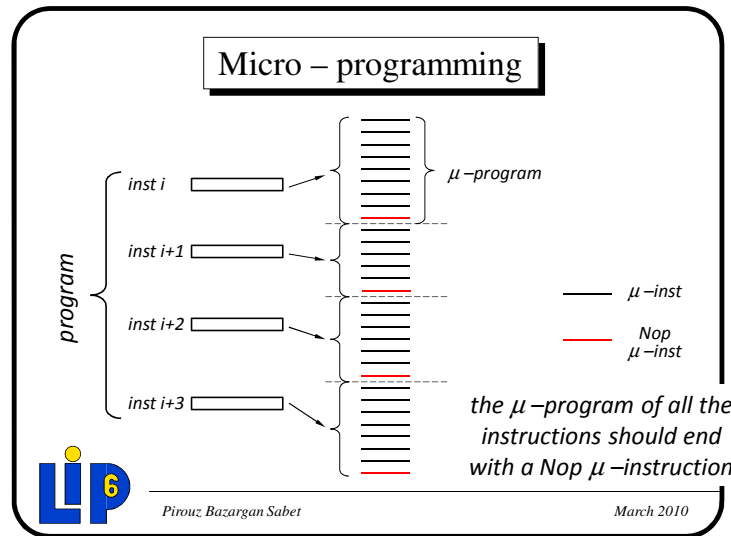
## Micro – programming

*Addu rd, rs, rt*



Pirouz Bazargan Sabet

March 2010



### Micro – programming

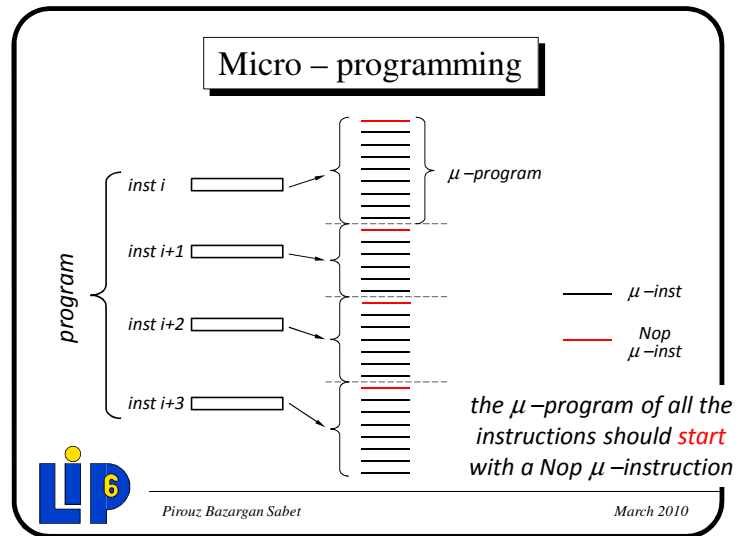
*Addu rd, rs, rt*

OPX	OPY	ALU	RES	MEM	EXC	SEQ
Npc	C4	Add	Npc	Nop	Nop	Opc
Rt	C0	Add	Dad	Nop	Nop	Jmp
Rs	Dad	Add	Rd	Nop	Nop	Jmp
C0	C0	Add	Nop	Rdi	Nop	Jmp

**LIP 6**

Pirouz Bazargan Sabet

March 2010



### Micro – programming

*Addu rd, rs, rt*

OPX	OPY	ALU	RES	MEM	EXC	SEQ
Npc	C4	Add	Npc	Nop	Nop	Opc
Rt	C0	Add	Dad	Nop	Nop	Fun
Rs	Dad	Add	Rd	Nop	Nop	Jmp
C0	C0	Add	Nop	Rdi	Nop	Jmp

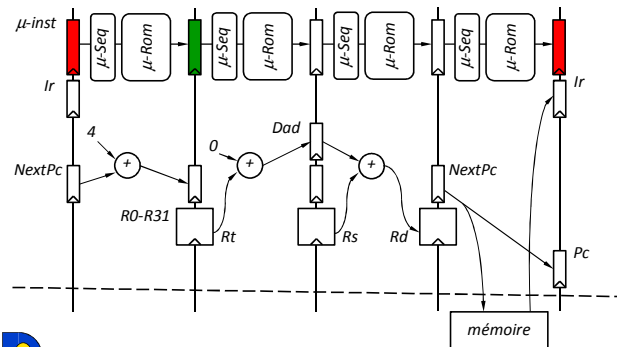
**LIP 6**

Pirouz Bazargan Sabet

March 2010

## Micro – programming

*Addu rd, rs, rt*



Pirouz Bazargan Sabet

March 2010

## Outline

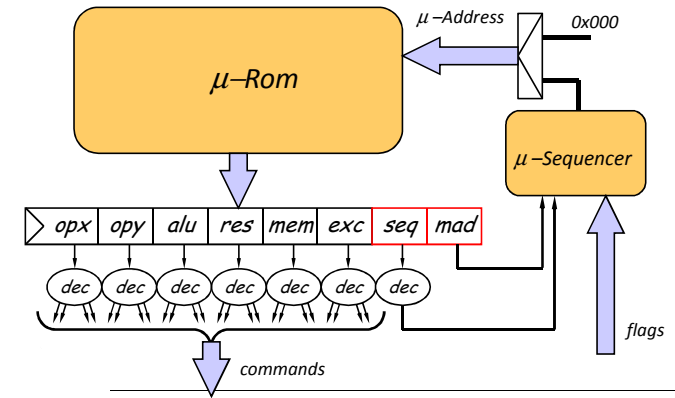
- Introduction
- Architecture
- Implementation



Pirouz Bazargan Sabet

March 2010

## Implementation- Reset



Pirouz Bazargan Sabet

March 2010

## Reset

**Reset** Abort the current program  
Jump to the Reset Handler

- Initialize the address of the next instruction : **0xBF00 0000**
- Initialize the Status Register : **0x0040 0004**
- Initialize the Cause Register : **0x0000 0000**
- Save the return address into Eepc
- Initialize the Exception Base Register : **0x8000 0000**



Pirouz Bazargan Sabet

March 2010

## Interrupt

**Interrupt** Stop executing the current program  
Execute the Interrupt Handler  
Resume the interrupted program

- Initialize the address of the next instruction :  
if  $BootExcVect = 0$  **EXH-ADR**  
if  $BootExcVect = 1$  **0xBF00 0380**
- Save the return address in EPC
- Set the SR : Mode (Kernel), set the Global Int Mask
- Set the Cause



Pirouz Bazargan Sabet

March 2010

## Micro – programming

*Addu rd, rs, rt*

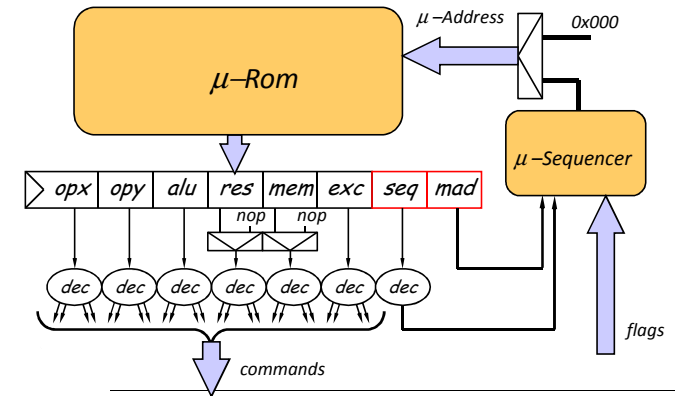
OPX	OPY	ALU	RES	MEM	EXC	SEQ
Npc	C4	Add	Npc	Nop	Nop	Opc
Rt	C0	Add	Dad	Nop	Nop	Fun
Rs	Dad	Add	Rd	Nop	Nop	Jump
C0	C0	Add	Nop	Rdi	Nop	Exc



Pirouz Bazargan Sabet

March 2010

## Implementation- Reset



Pirouz Bazargan Sabet

March 2010

## Micro – programming

*Addu rd, rs, rt*

OPX	OPY	ALU	RES	MEM	EXC	SEQ
Npc	C4	Add	Npc	Nop	Nop	Opc
Rt	C0	Add	Dad	Nop	Nop	Fun
C0	C0	Add	Nop	Nop	lia	Jump
Rs	Dad	Add	Rd	Nop	Nop	Jump
C0	C0	Add	Nop	Rdi	Nop	Exc



Pirouz Bazargan Sabet

March 2010

## Micro – programming

*Addu rd, rs, rt*

OPX	OPY	ALU	RES	MEM	EXC	SEQ
Npc	C4	Add	Npc	Nop	Nop	Opc
Rt	C0	Add	Dad	Nop	Nop	Fun
C0	C0	Add	Nop	Nop	lia	Jump
Rs	Dad	Add	Rd	Nop	Nop	Jump
C0	C0	Add	Nop	Rdi	lbe	Exc



Pirouz Bazargan Sabet

March 2010