Technical University of Cluj-Napoca
Computer Science Department

Computer Architecture

Lecturer: Mihai Negru

2nd Year, Computer Science

Lecture 3: MIPS ISA

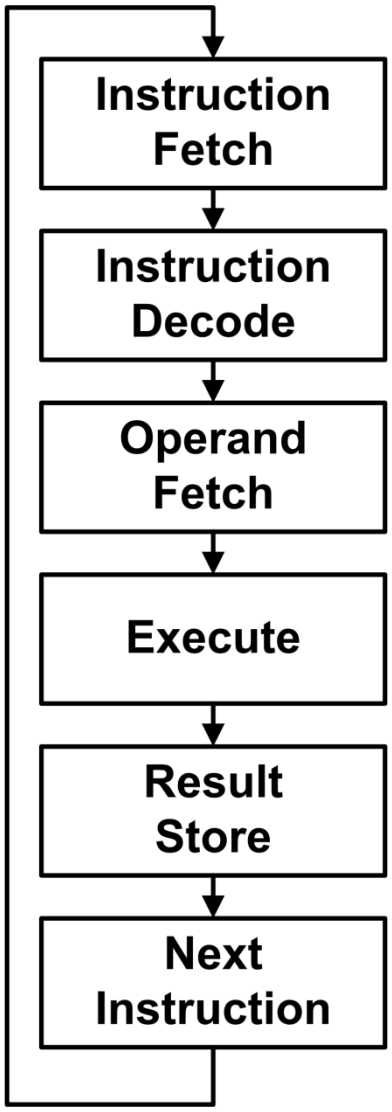
http://users.utcluj.ro/~negrum/

MIPS ISA

- MIPS – **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages

- ISA – Instruction Set Architecture
  - The interface between Hardware and Software
  - ISA Components:
    - Memory organization
    - Registers
    - Data Types and Data Structures
    - Instruction Formats
    - Instruction Set
    - Addressing modes
    - Flow of Control
    - Input/Output
    - Interrupts
    - …

Instruction Execution Cycle



Obtain/Read Instruction from Program Memory/Storage

Determine the instruction format or encoding

Locate and obtain the operands of the instruction and the location of the result

Compute result of the instruction:

- value, status or address

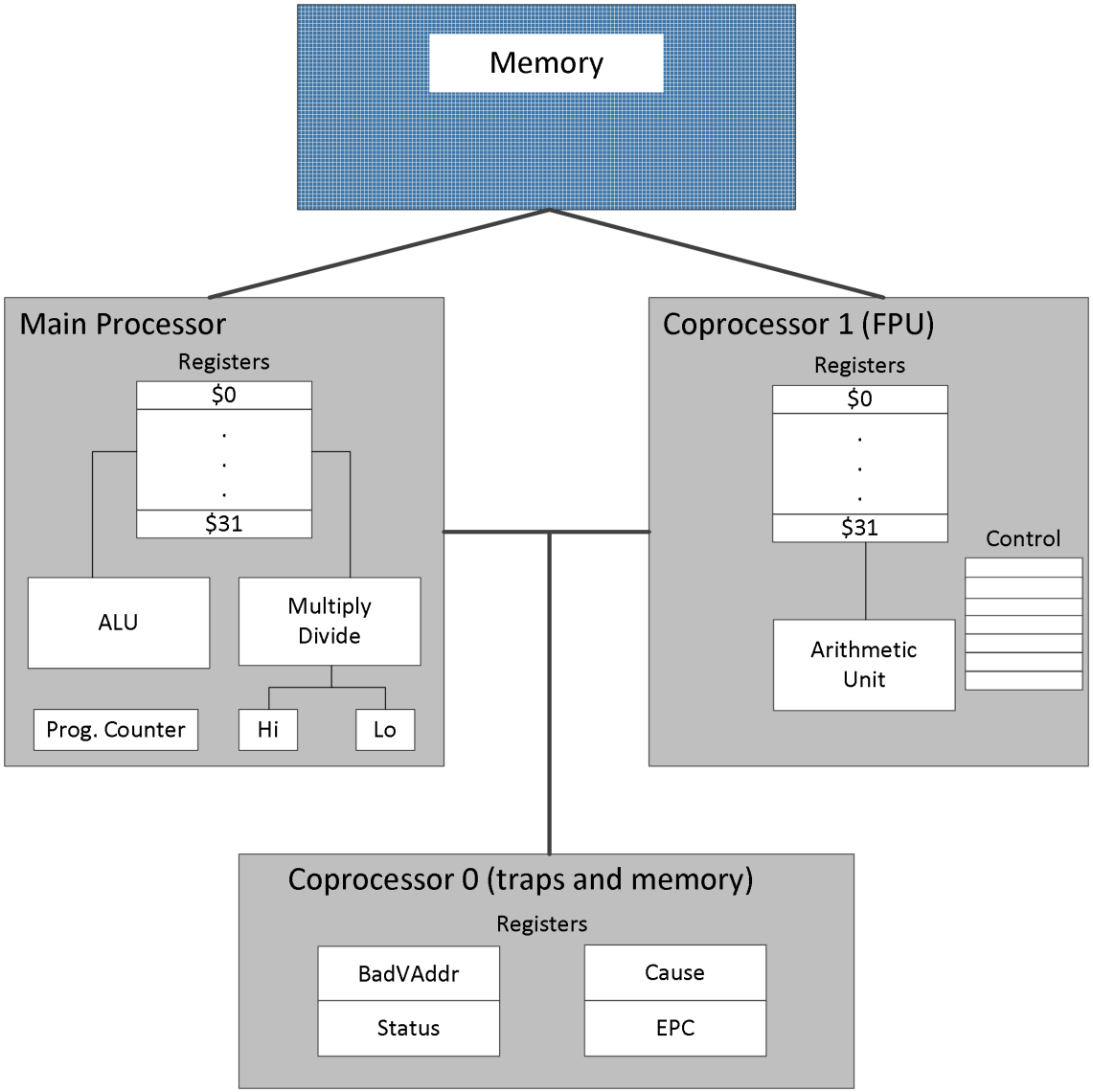
Write/Save the result of the instruction:

- register or memory

Determine the next instruction (address) to be executed:

- Normal operation
- Jumps, Conditions, Branches

MIPS Architecture – Block Diagram



MIPS – LOAD/STORE ISA

- 3 Address Machine
- General Purpose Register
- 32 bit µprocessor

- Main Processor – CPU
- Coprocessor 0 – OS Support
- Coprocessor 1 – FPU
- Memory Unit

MIPS CPU Registers

- 32 x 32-bit General purpose registers
  - R0 – R31
  - R0 has fixed value of zero.
    - Attempt to writing into R0 is not illegal, but it is ineffective
  - R31 holds the return address of a procedure call

- PC – Program Counter
  - contains the address of the next instruction to be fetched

- Hi, Lo
  - store partial result of multiplication and division operations

MIPS General Purpose Registers

Name	Register Number	Usage
$zero	0	Always 0
$at	1	Reserved for assembler
$v0 - $v1	2-3	Expression evaluation and function results
$a0 - $a3	4-7	Arguments
$t0 - $t7	8-15	Temporary, saved by caller
$s0 - $s7	16-23	Temporary, saved by called function
$t8 - $t9	24-25	Temporary, saved by caller
$k0 - $k1	26-27	Reserved for kernel (OS)
$gp	28	Global Pointer
$sp	29	Stack Pointer (top of stack)
$fp	30	Frame Pointer (beginning of current frame)
$ra	31	Return Address (pushed by call instruction)

MIPS Coprocessors

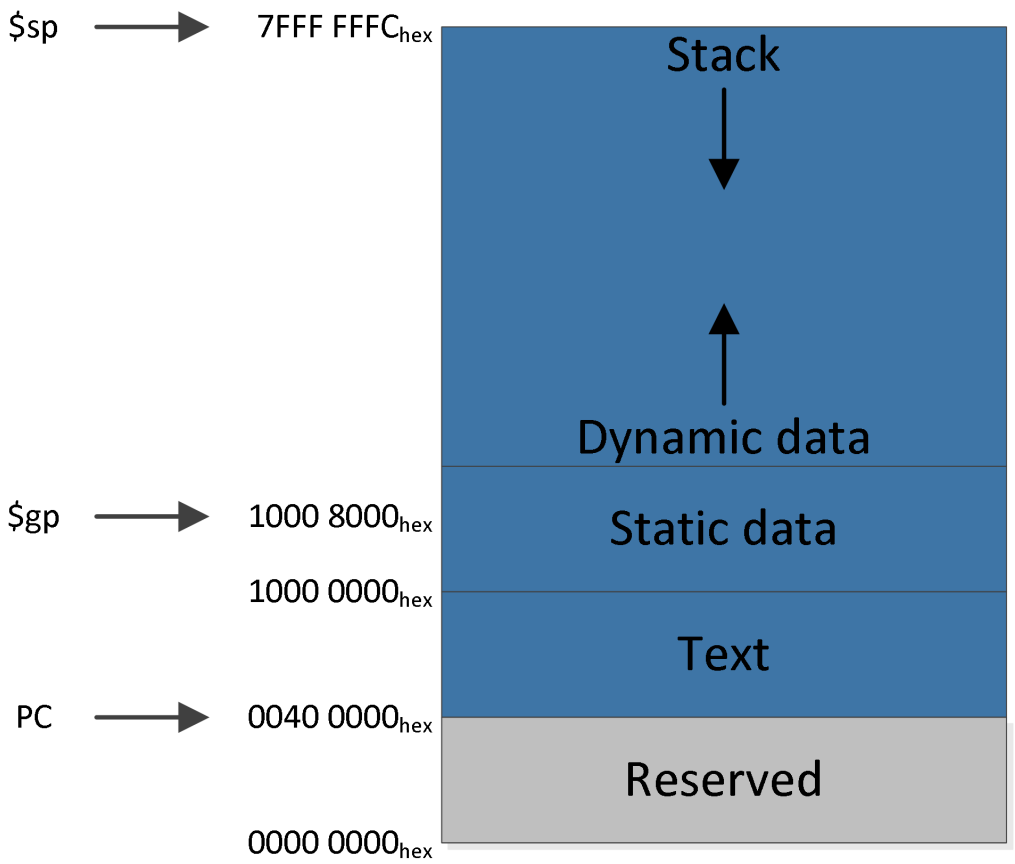
- Coprocessor 0 – CP0
  - Provides support for the operating system (OS)
    - Exception handling
    - Memory management
    - Scheduling and control of critical resources
  - CP0 registers – 16 registers
    - Status Register (CP0, Reg12) – processor status and control, interrupt bits
    - Cause Register (CP0, Reg13)– cause of the most recent interrupt
    - EPC Register (CP0, Reg14) – program counter at the last interrupt
    - BadVAddr Register (CP0, Reg08) – virtual address for the most recent address related exception
  - Instructions for reading and writing the CP0 Registers:
    - mfc0 $k0, $13 # Move from CP0, Reg13 (Cause Register) in $k0
    - mtc0 $0, $12 # Move value 0 in CP0, Reg12 (Status Register)
- Coprocessor 1 – CP1, Floating Point Unit (FPU)
  - 32x32-bit Floating Point Registers (f0 – f31)
  - Five control registers

MIPS Memory Organization

- A large, single-dimension array
  - A memory address – an index in the array
- Byte addressing – the index points to a byte of memory
- Word – 32 bits (4 bytes)
- 32-bit addresses
  - 2^{32} bytes → 2^{30} words:

0, 1, 2, … to 2^{32}-1 → 0, 4, 8, …to 2^{32}-4

- Endianness – can be selected
  - Little / Big endian
- Data Alignment: YES
  - 32 bit word starts at a multiple of 4 bytes address
  - 16 bit word starts at multiple of 2 bytes address

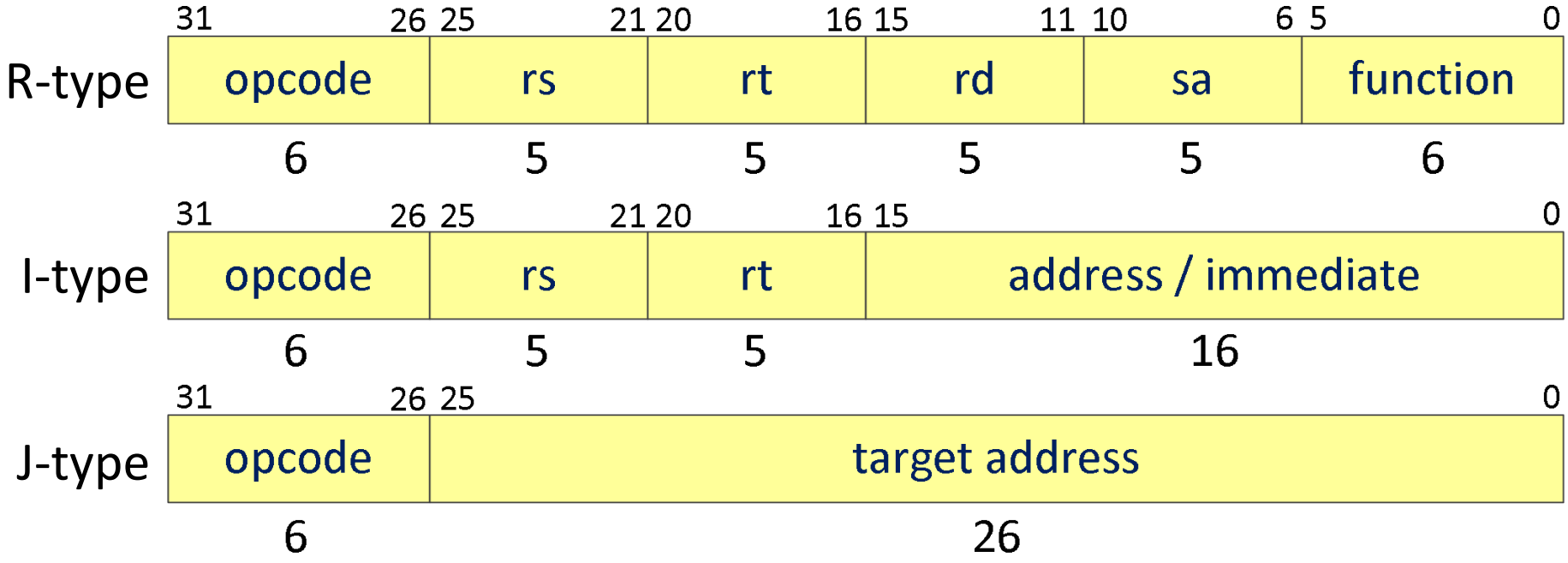


MIPS Data Types

- MIPS operates on:
  - 32-bit (unsigned or 2's complement) integers
  - 32-bit (single precision floating point) real numbers
  - 64-bit (double precision floating point) real numbers
- Locations in General Purpose Registers
  - 32-bit words, bytes (8-bits) and half words (16-bits) can be loaded into GPRs
  - When loading into GPRs, bytes and half words are Zero or Sign Extended to fill the 32 bits
- Locations in Floating Point Registers
  - Only 32-bit units can be loaded into FPRs
  - 32-bit real numbers are stored in FPRs
  - 64-bit real numbers are stored in two consecutive FPRs, starting with even numbered register

MIPS Instruction Formats

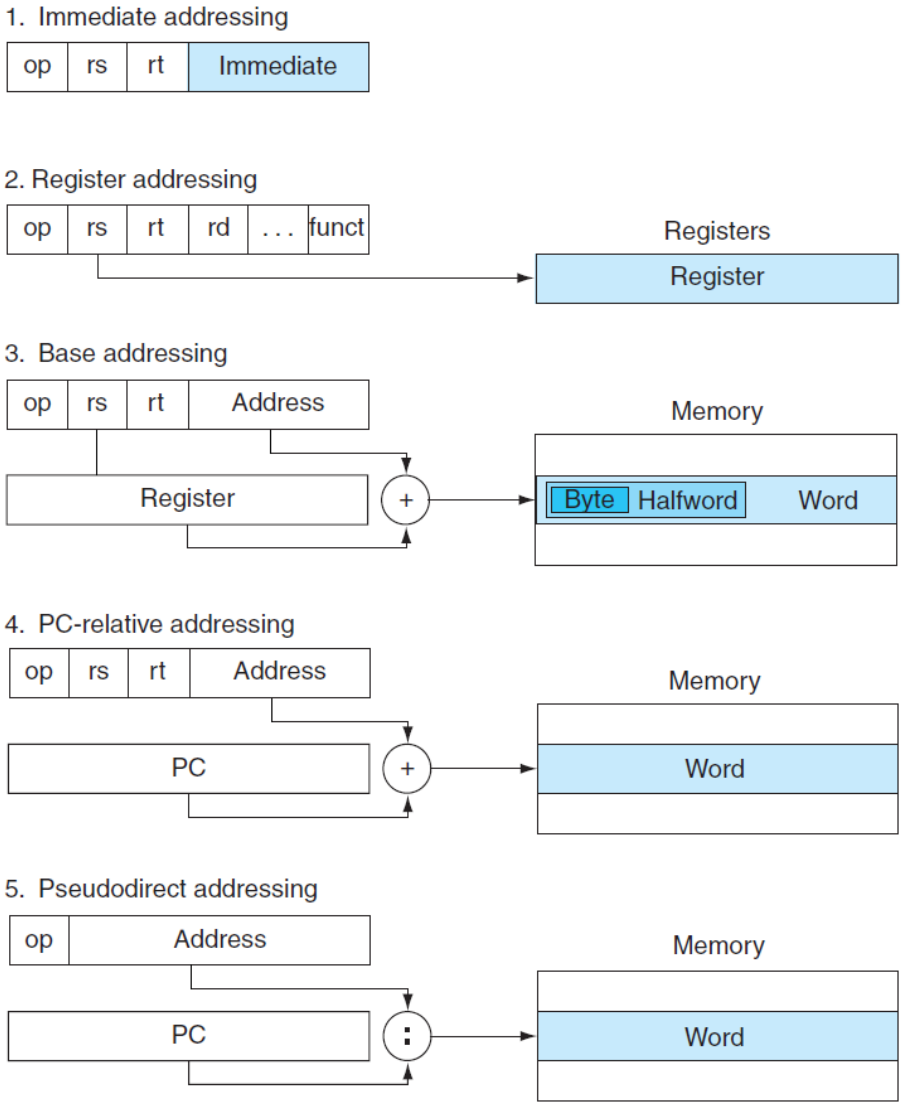
- 32-bit length instructions
- 3 instruction formats
  - R-type instructions – used for arithmetical/logical operations
  - I-type instructions – used for arithmetical/logical operations with immediate values, memory data transfers and conditional jumps or branches
  - J-type instructions – used for unconditional jumps



MIPS Instruction Formats

Field	Description
opcode	6-bit primary operation code
rs	5-bit source register specifier
rt	5-bit target (source/destination) register specifier or used to specify functions within the primary opcode value REGIMM
rd	5-bit destination register specifier
sa	5-bit shift amount
function	6-bit function field used to specify functions within the primary opcode SPECIAL (000000)
address / immediate	16-bit immediate used for: logical operands, arithmetic signed operands
	load/store address byte offsets
	PC-relative branch signed instruction displacement
target address	26-bit index shifted left two bits to supply the low-order 28 bits of the jump target address

MIPS Addressing Modes



[1]

- Immediate addressing
- Register addressing
- Base addressing (indexed)
  - The only memory data addressing
  - Register content + address
  - r_0 + Address → absolute addressing
  - r_i + Address (=0) → register indirect
- PC-relative addressing
  - Branch instructions
  - Branch address: PC + 4 + 4*Address
- Pseudo-direct addressing
  - Jump Instructions
  - Jump address: PC[31:28]||4*Address
  - Jumps in 256 MB regions

MIPS Core Instruction Set

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	Three operands
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	+ constant
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load half unsigned	lhu $s1,100($s2)	$s1 = Memory[$s2 + 100]	Halfword memory to register
	store half	sh $s1,100($s2)	Memory[$s2 + 100] = $s1	Halfword register to memory
	load byte unsigned	lbu $s1,100($s2)	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	sb $s1,100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui $s1,100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 |$s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; two's complement
	set less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; two's complement
	set less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; unsigned numbers
	set less than immediate unsigned	sltiu $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; unsigned numbers
Uncondi-tional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

[1]

MIPS Instructions

Assembly	Type	RTL Abstract	Instruction
add $rd, $rs, $rt	R	RF[rd] ← RF[rs] + RF[rt]	Add
sub $rd, $rs, $rt	R	RF[rd] ← RF[rs] – RF[rt]	Subtract
addi $rt, $rs, imm	I	RF[rt] ← RF[rs] + S_Ext(imm)	Add Immediate
addu $rd, $rs, $rt	R	RF[rd] ← RF[rs] + RF[rt]	Add Unsigned (no overflow)
subu $rd, $rs, $rt	R	RF[rd] ← RF[rs] – RF[rt]	Subtract Unsigned
mult $rs, $rt	R	Hi,Lo ← RF[rs] * RF[rt]	Signed Multiplication
multu $rs, $rt	R	Hi,Lo ← RF[rs] * RF[rt]	Unsigned Multiplication
div $rs, $rt	R	Lo ← RF[rs] / RF[rt], Hi ← RF[rs] mod RF[rt]	Signed Division
divu $rs, $rt	R	Lo ← RF[rs] / RF[rt], Hi ← RF[rs] mod RF[rt]	Unsigned Division
mfhi $rd	R	RF[rd] ← Hi	Move from Hi
mflo $rd	R	RF[rd] ← Lo	Move from Lo
		…	

MIPS Instructions

Assembly	Type	RTL Abstract	Instruction
and $rd, $rs, $rt	R	RF[rd] ← RF[rs] & RF[rt]	Logical AND
or $rd, $rs, $rt	R	RF[rd] ← RF[rs] | RF[rt]	Logical OR
xor $rd, $rs, $rt	R	RF[rd] ← RF[rs] ^ RF[rt]	Exclusive OR
andi $rt, $rs, imm	I	RF[rt] ← RF[rs] & Z_Ext(imm)	Logical AND unsigned constant
ori $rt, $rs, imm	I	RF[rt] ← RF[rs] | Z_Ext(imm)	Logical OR unsigned constant
xori $rt, $rs, imm	I	RF[rt] ← RF[rs] ^ Z_Ext(imm)	Exclusive OR unsigned constant
sll $rd, $rt, sa	R	RF[rd] ← RF[rt] << sa	Shift Left Logical
srl $rd, $rt, sa	R	RF[rd] ← RF[rt] >> sa	Shift Right Logical
sra $rd, $rt, sa	R	RF[rd] ← RF[rt] >> sa	Shift Right Arithmetic
sllv $rd, $rs, $rt	R	RF[rd] ← RF[rs] << RF[rt]	Shift Left Logical Variable
srlv $rd, $rs, $rt	R	RF[rd] ← RF[rs] >> RF[rt]	Shift Right Logical Variable
srav $rd, $rs, $rt	R	RF[rd] ← RF[rs] >> RF[rt]	Shift Right Arithmetic Variable
		…	

MIPS Instructions

Assembly	Type	RTL Abstract	Instruction
lw $rt, imm($rs)	I	RF[rt] ← M[RF[rs] + S_Ext(imm)]	Load Word
sw $rt, imm($rs)	I	M[RF[rs] + S_Ext(imm)] ← RF[rt]	Store Word
lb $rt, imm($rs)	I	RF[rt] ← S_Ext(M[RF[rs] + S_Ext(imm)])	Load Byte
sb $rt, imm($rs)	I	M[RF[rs] + S_Ext(imm)] ← S_Ext(RF[rt])	Store Byte
lui $rt, imm	I	RF[rt] ← imm || 0x0000	Load Upper Immediate
beq $rt, $rs, imm	I	If(RF[rs] == RF[rt]) then PC ← PC + 4 + S_Ext(imm) <<2	Branch on equal
bne $rs, $rt, imm	I	If(RF[rs]!=RF[rt]) then PC ← PC + 4 + S_Ext(imm) <<2	Branch on not equal
bgez $rs, imm	I	If(RF[rs] >= 0]) then PC ← PC + 4 + S_Ext(imm) <<2	Branch on Greater Than or Equal to Zero
bltz $rs, imm	I	If(RF[rs] < 0]) then PC ← PC + 4 + S_Ext(imm) <<2	Branch on Less Than Zero
		…	

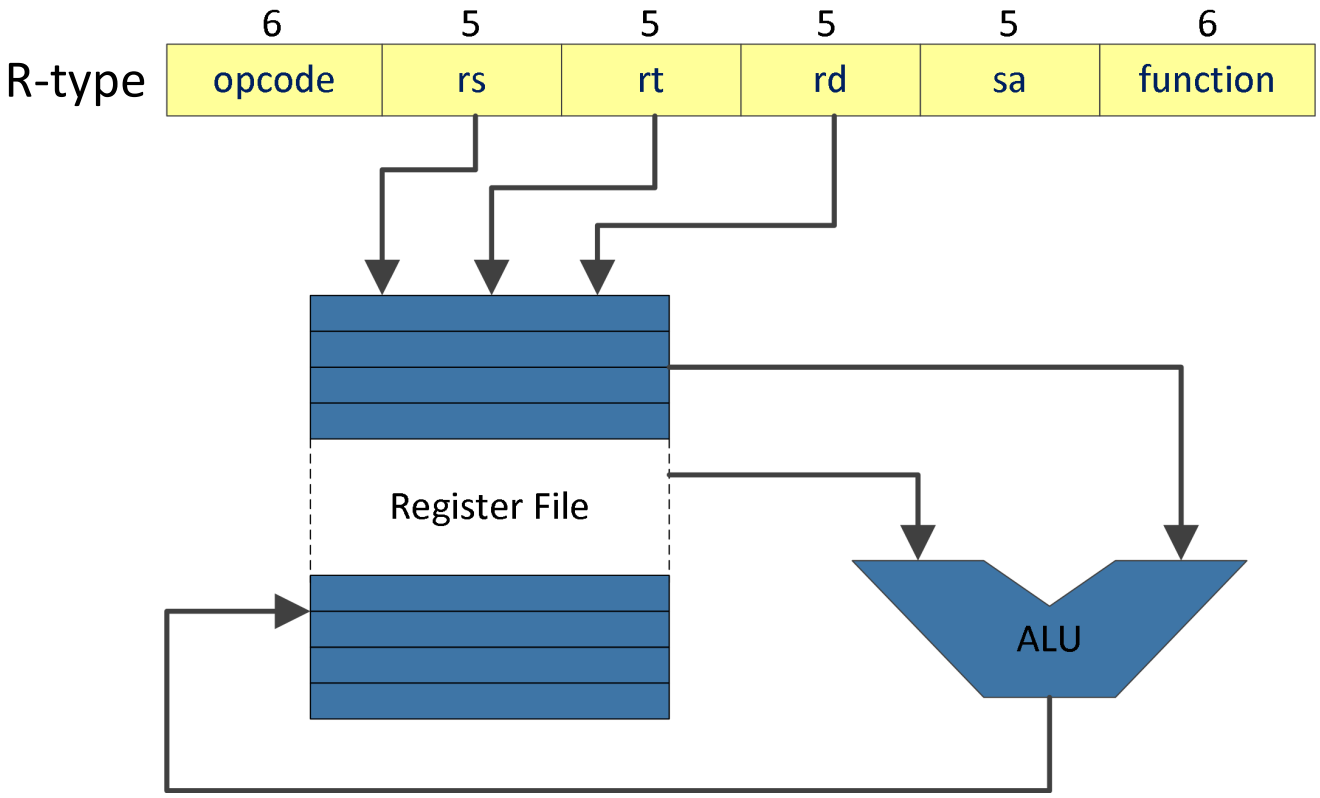
MIPS Instructions

Assembly	Type	RTL Abstract	Instruction
slt $rd, $rs, $rt	R	If(RF[rs] < RF[rt]) then RF[rd] ←1 else RF[rd] ← 0	Set on Less Than
slti $rt, $rs, imm	I	If(RF[rs] < S_Ext(imm)) then RF[rt] ← 1 else RF[rt] ← 0	Set on Less Than Immediate
sltu $rd, $rs, $rt	R	If(RF[rs] < RF[rt]) then RF[rd] ← 1 else RF[rd] ← 0	Set on Less Than Unsigned
sltiu $rt, $rs, imm	I	If(RF[rs] < Z_Ext(imm)) then RF[rt] ← 1 else RF[rt] ← 0	Set on Less Than Immediate Unsigned
j target_address	J	PC ← PC[31:28] || target_address || 00	Jump
jal target_address	J	RF[31] ← PC + 4 PC ← PC[31:28] || target_address || 00	Jump And Link RF[31] – return address
jr $rs	R	PC ← RF[rs]	Jump Register
		…	

MIPS Instructions

- Arithmetic / Logical R-type Instructions

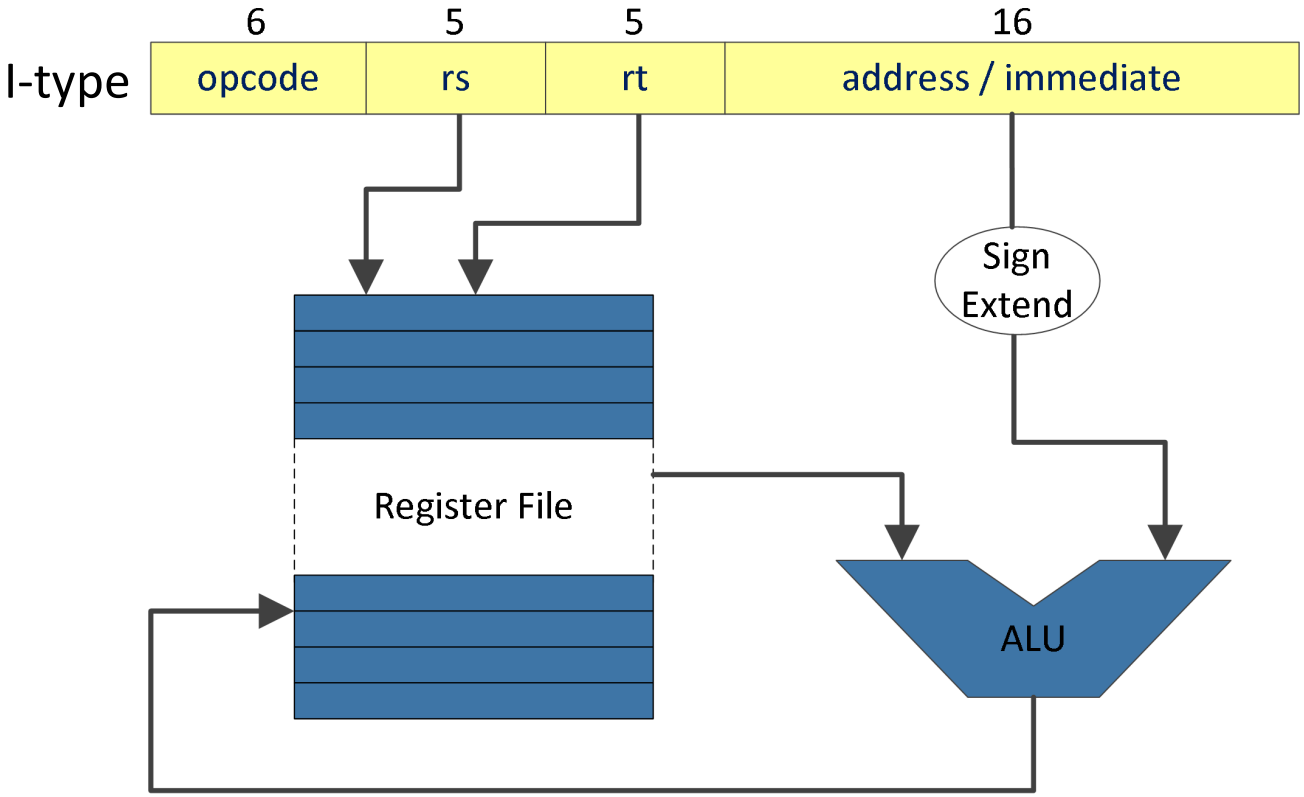
add $rd, $rs, $rt → RF[rd] ← RF[rs] + RF[rt]



MIPS Instructions

- Immediate Arithmetic Instructions

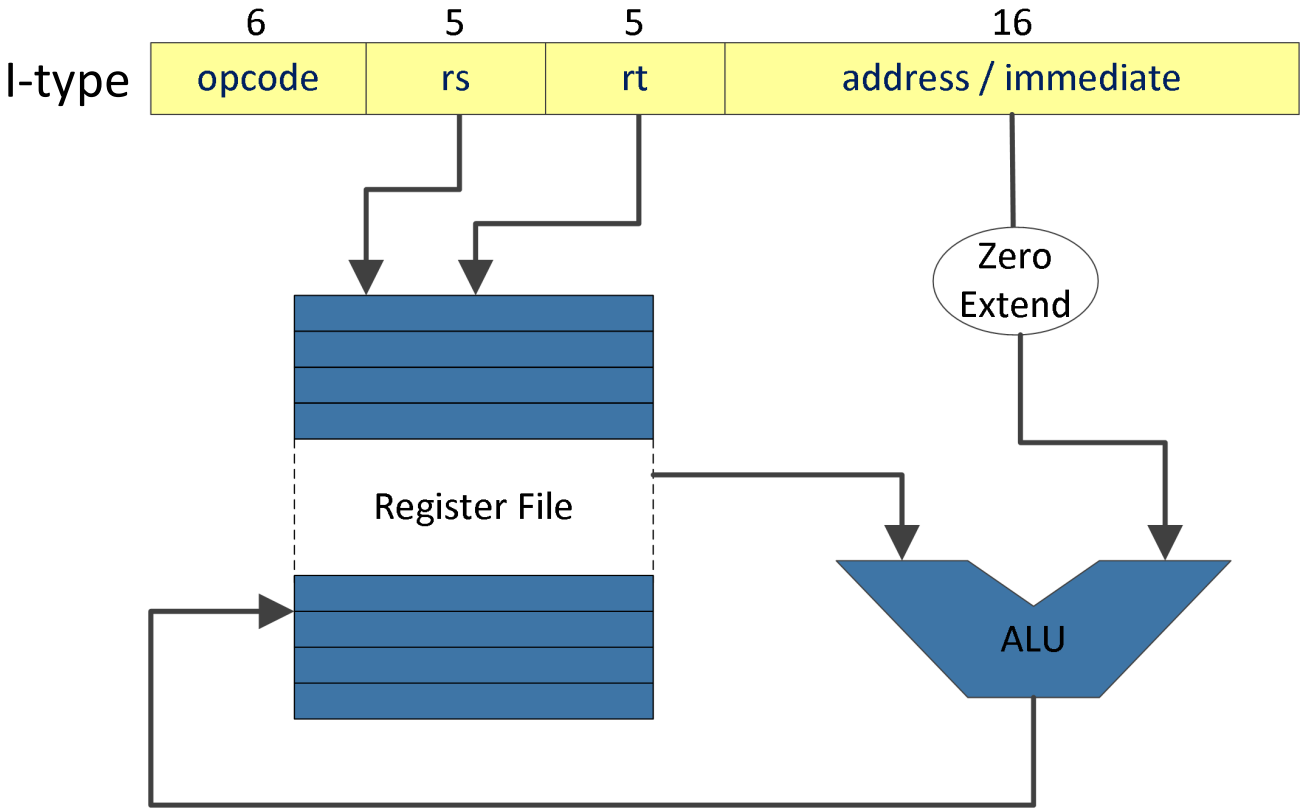
addi $rt, $rs, imm → RF[rt] ← RF[rs] + S_Ext(imm)



MIPS Instructions

- Immediate Logical Instructions

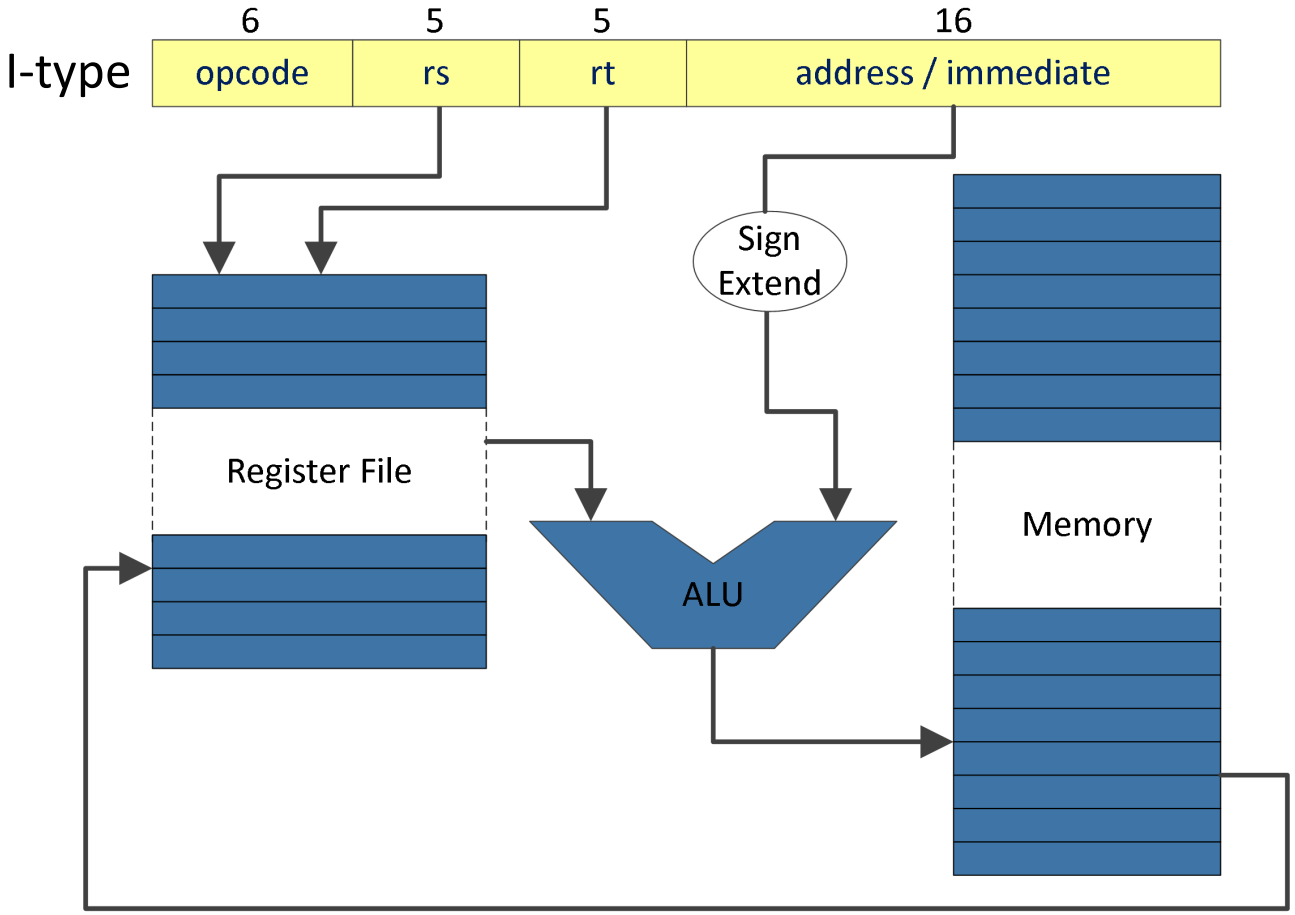
  andi $rt, $rs, imm → RF[rt] ← RF[rs] & Z_Ext(imm)



MIPS Instructions

- Load Word Instruction

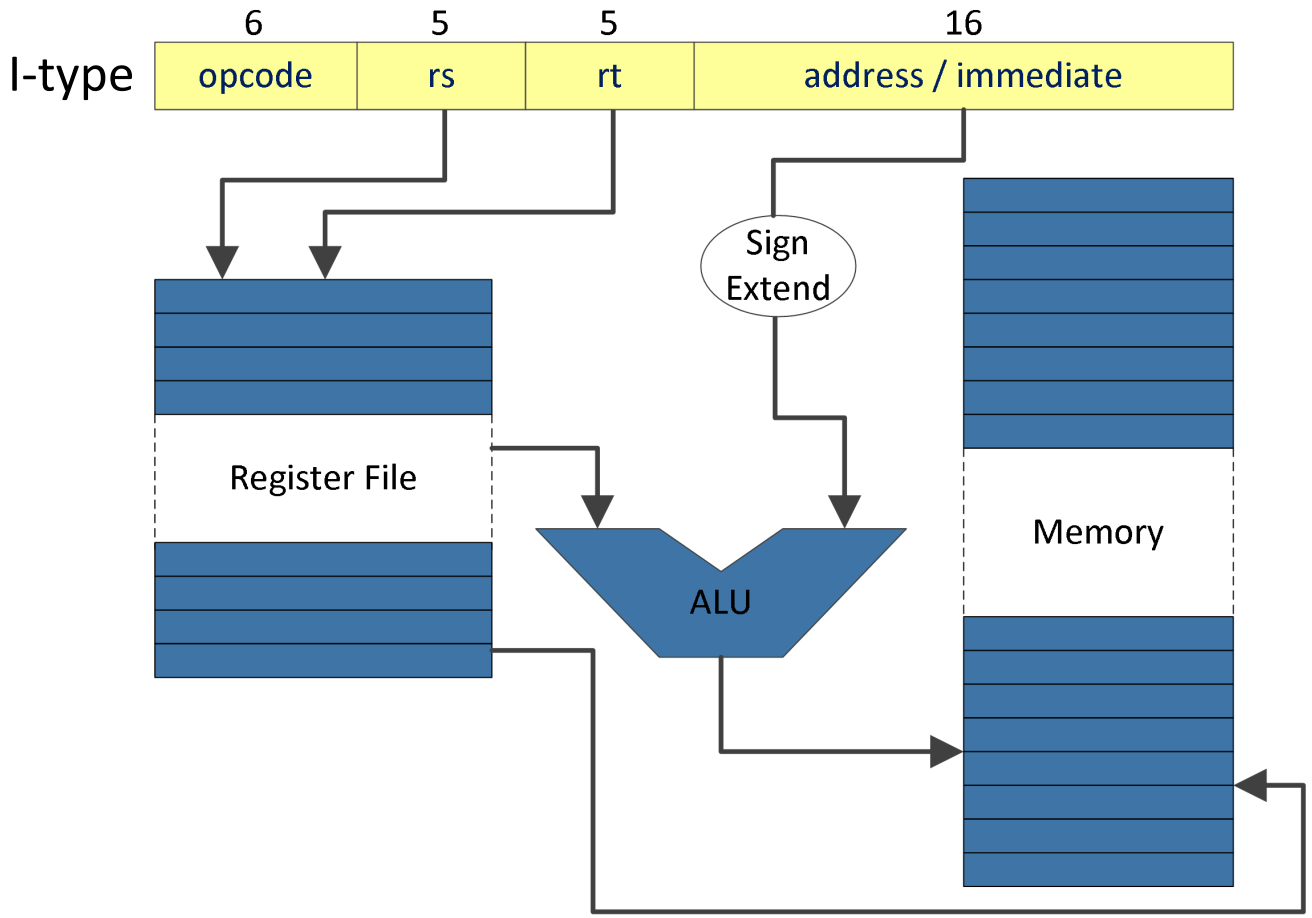
lw $rt, imm($rs) → RF[rt] ← M[RF[rs] + S_Ext(imm)]



MIPS Instructions

- Store Word Instruction

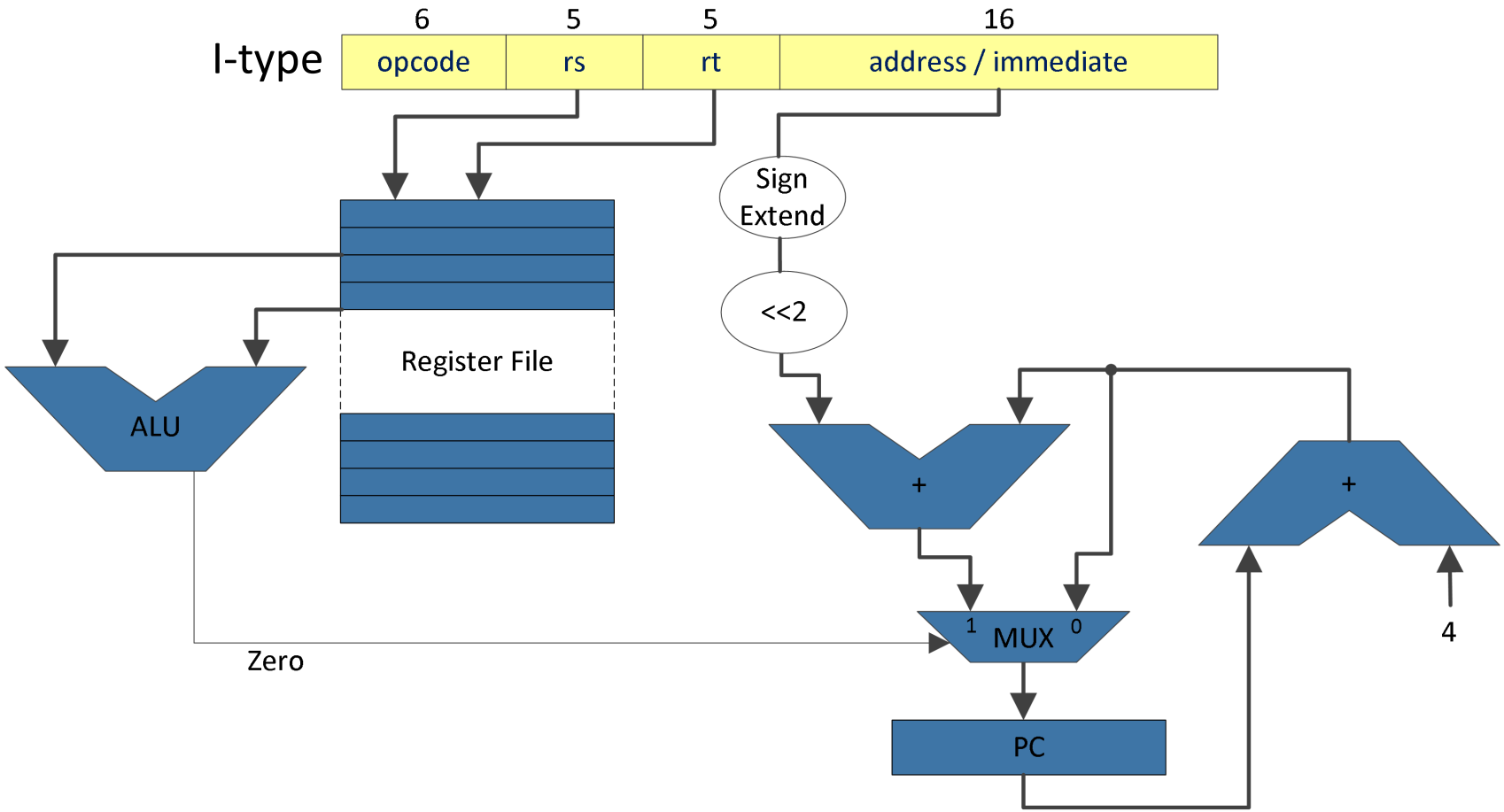
sw $rt, imm($rs) → M[RF[rs] + S_Ext(imm)] ← RF[rt]



MIPS Instructions

- Branch on Equal Instruction

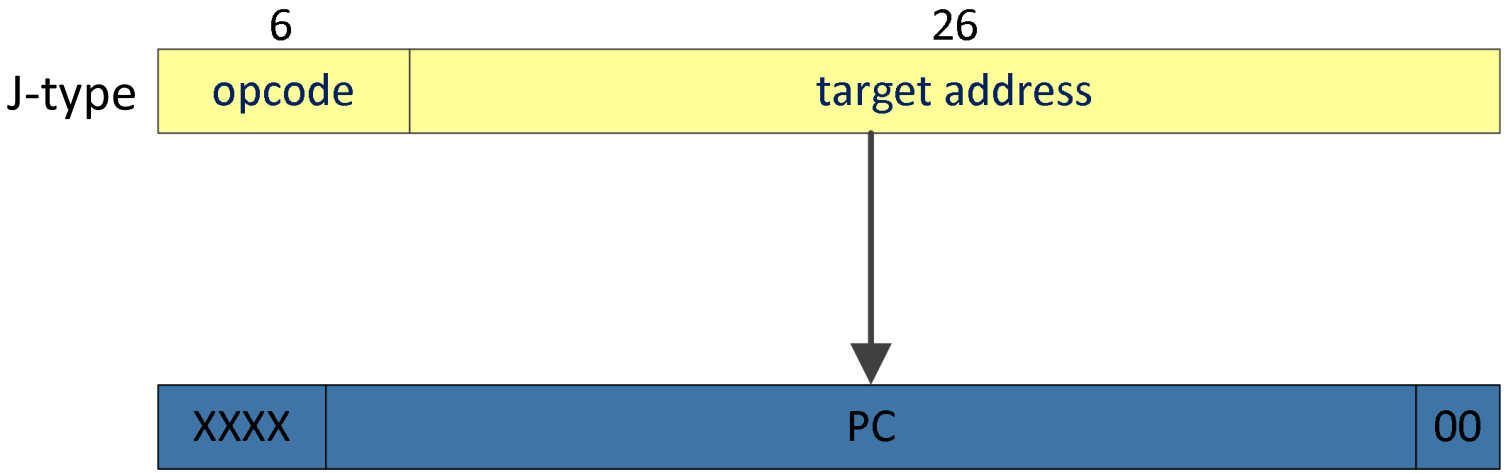
beq $rt, $rs, imm→If(RF[rs]==RF[rt]) then PC ← PC+4+S_Ext(imm)<<2



MIPS Instructions

- Jump Instruction

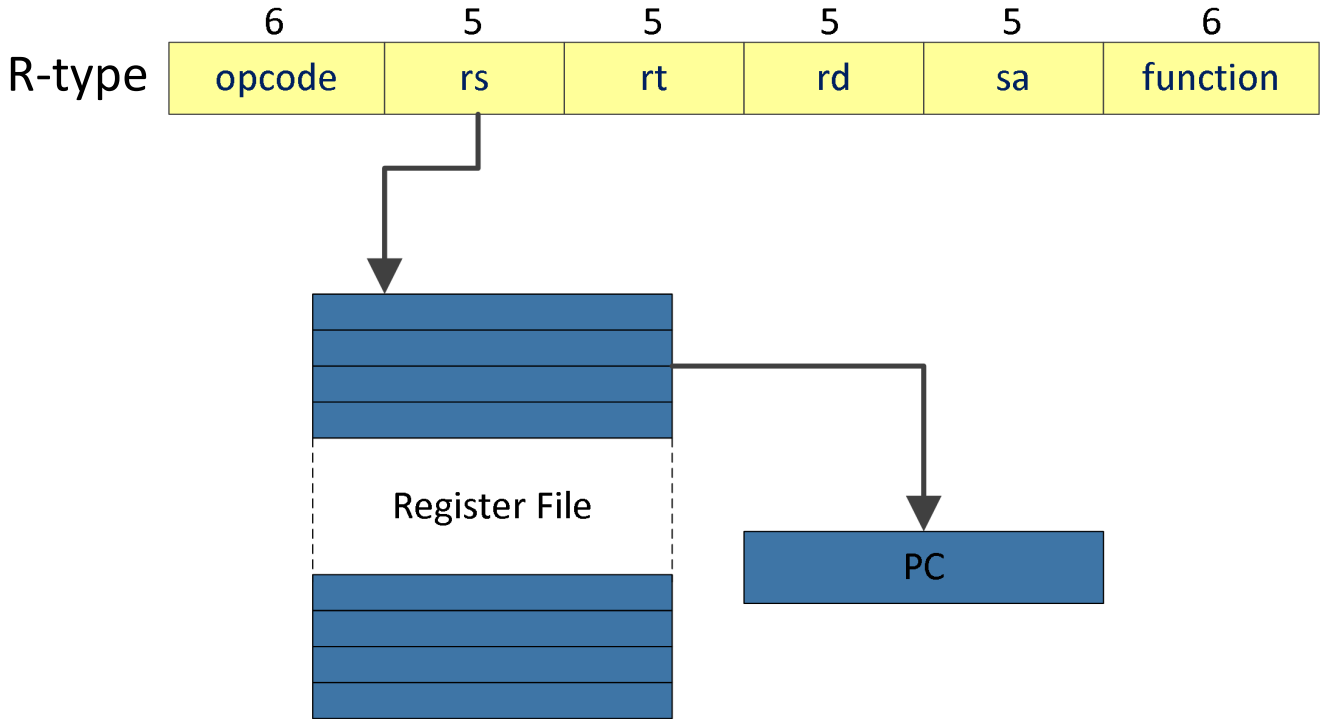
j target_address → PC ← PC[31:28] || target_address || 00



MIPS Instructions

- Jump Register Instruction

jr $rs → PC ← RF[rs]



Pseudo Instructions

- Not accepted by the processor
- Can be accepted by compiler and implemented by native processor instructions
- Examples:

Absolute Value		

abs $rd, $rs	addu $rd, $0, $rs
	bgez $rs, 1
	sub $rd, $0, $rs
		slt $ra, $rs, $0
		bne $ra, $0, NEG
		add $rd, $rs, $0
		j DONE
		NEG: sub $rd, $0, $rs
		DONE:

Branch on Equal to Zero		
beqz $rs, label	beq $rs, $0, label	

Pseudo Instructions

Branch on Greater than or Equal		Move	
bge $rs, $rt, label	slt $at, $rs, $rt beq $at, $0, label	move $rd,$rs	addu $rd, $0, $rs
Branch on Greater than or Equal Unsigned		**Negate**	
bgeu $rs, $rt, label	sltu $at, $rs, $rt beq $at, $0, label	neg $rd, $rs	sub $rd, $0, $rs
Branch if Greater Than		**No Operation**	
bgt $rs, $rt, label	slt $at, $rt, $rs bne $at, $0, label	nop	or $0, $0, $0
		nop	sll $0, $0, $0
Load Address			
la $rd, label	lui $at, Upper 16-bits of label ori Rd, $at, Lower 16-bits of label		
Load Immediate			
li $rt, value	ori $rt, $0, value		

MIPS Interrupt Processing

- MIPS hardware interrupt processing → 3 steps
- Step 1: Save PC
  - EPC (Exception Program counter) gets a value equal to:
    - The address of the instruction that generates an exception (address error, reserved instruction) or hardware malfunction detected (memory parity error)
    - The address of the next instruction: external interrupts
- Step 2: PC gets new value and interrupt cause code is saved
  - PC ← $8000\ 0180_{16}$ (the address depends on MIPS)
  - Cause Register ← interrupt code
  - Each interrupt has its code, e.g.:
    - hardware interrupt = 0
    - illegal memory address (load/fetch or store) = 4 or 5
    - bus error (fetch or load/store)= 6 or 7
    - syscall instruction execution = 8
    - illegal op-code, i.e. reserved or undefined op-code= 10
    - integer overflow = 12
    - any floating point exception = 15
- Step3: CPU Mode operation set to kernel mode
  - CPU Mode bit ← 0

Problems – Homework

- What is the address space of a MIPS processor (what is the total memory size that it can address)
- Using real MIPS instructions, write a pseudo instruction that computes the following:
  - Absolute value.
  - Branch on Greater Than
  - Branch on Less or Equal
  - Swap two registers
- Define new MIPS instructions:
  - LWR (load word register) – sums two registers to obtain the memory address.
  - SWR (store word register) – sums two registers to obtain the memory address.
  - LWA – uses a single register to obtain the memory address.
  - SWA – uses a single register to obtain the memory address.
  - Identify the instruction formats, write the RTL abstract and draw the processing diagram for each new instruction.

References

1. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 5th edition, ed. Morgan–Kaufmann, 2013.
2. D. A. Patterson and J. L. Hennessy, “Computer Organization and Design: A Quantitative Approach”, 5th edition, ed. Morgan-Kaufmann, 2011.
3. MIPS32™ Architecture for Programmers, Volume I: “Introduction to the MIPS32™ Architecture”.
4. MIPS32™ Architecture for Programmers Volume II: “The MIPS32™ Instruction Set”.