Arhitectura Calculatoarelor

Curs 3: MIPS: Arhitectura Setului de Instrucțiuni

E-mail: florin.oniga@cs.utcluj.ro

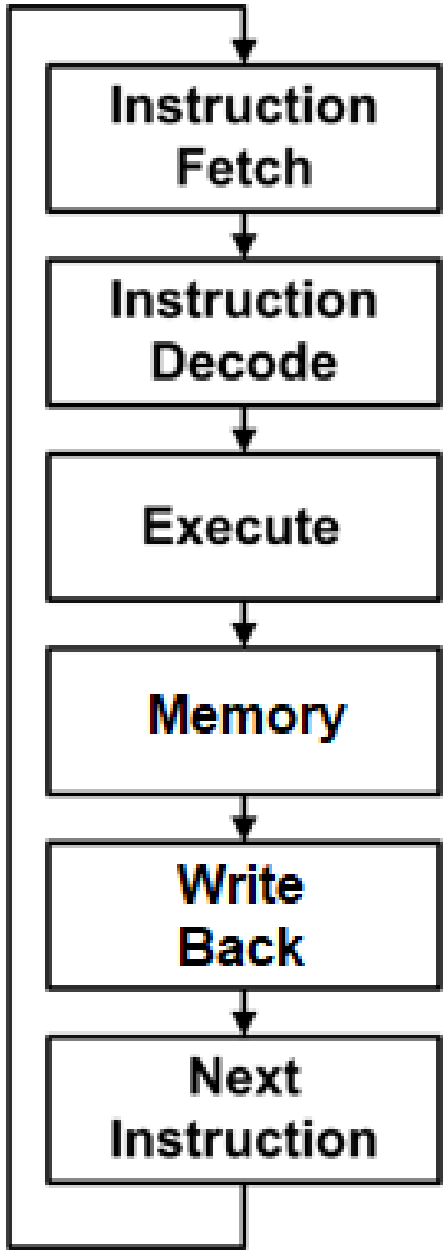
Web: http://users.utcluj.ro/~onigaf, secțiunea Teaching/AC

Arhitectura Setului de Instrucțiuni - MIPS

- **MIPS – Microprocessor without Interlocked Pipeline Stages**

- **Arhitectura Setului de Instrucțiuni MIPS – MIPS Instruction Set Architecture – ISA**
  - Este interfața între Hardware și Software
    - "Starea Vizibilă pentru programator": memoria, registre, contorul de program (program counter), etc.
  - Componente ISA:
    - Organizarea memoriei, Registre
    - Tipuri de date si Structuri de date: Codificări și reprezentări
    - Setul de instrucțiuni
    - Formate de instrucțiuni
    - Moduri de adresare; pentru instrucțiuni și date
    - Tratarea excepțiilor, protecții, I/O
    - Etc.

Instrucțiuni - Faze de execuție



Se extrage/aduce/citeşte instrucţiunea din memoria de program

Se decodifică instrucțiunea, citirea operanzilor tip registru

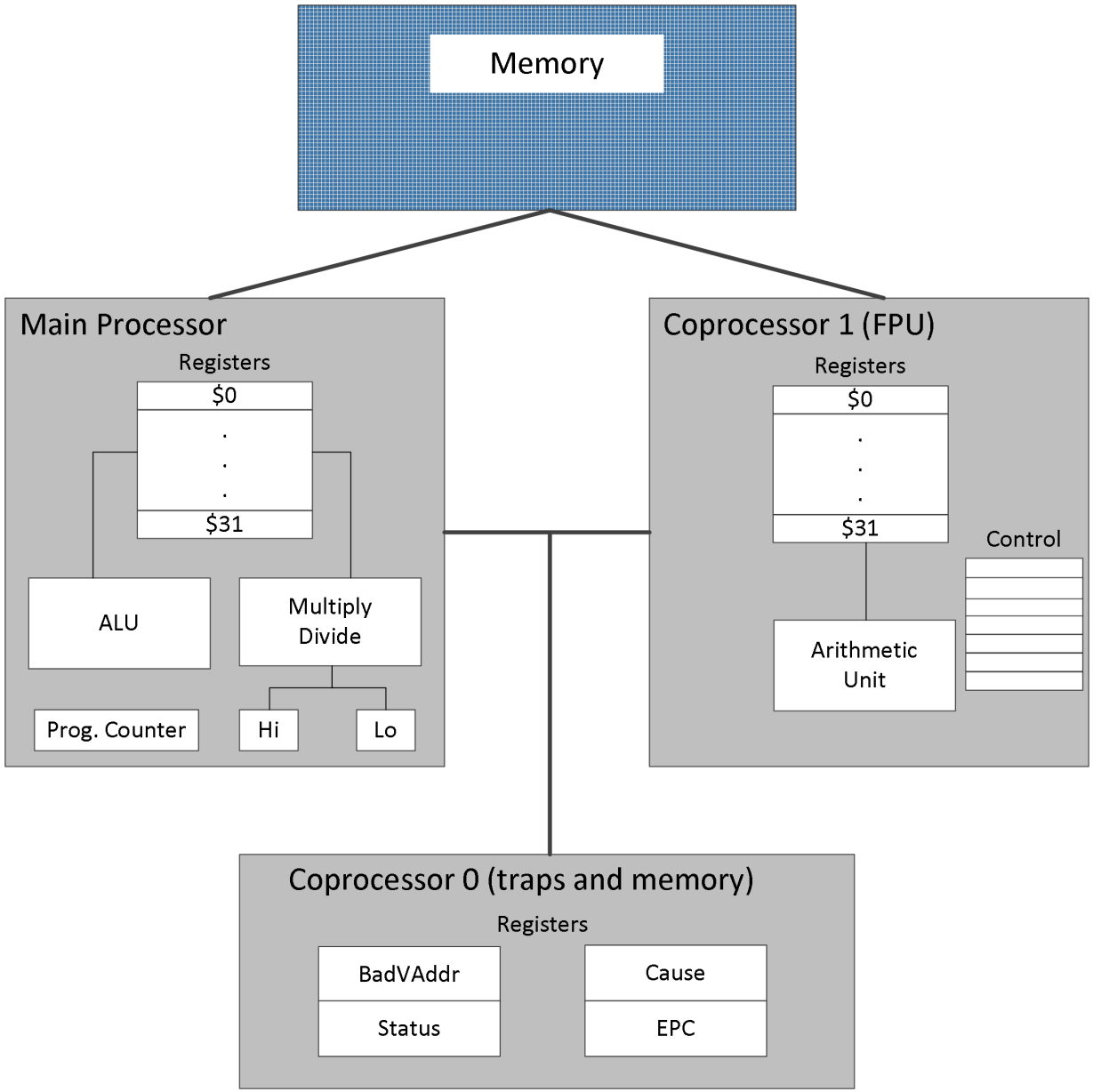
Se execută operaţia codificată

Se accesează memoria

Se memorează rezultatul

Se stabilește adresa instrucţiunii următoare (normal, salt, ramificare condiționată) – această fază nu se va mai prezenta explicit deoarece diferite calcule asociate sunt efectuate în celelalte faze!

Arhitectura MIPS – Schemă bloc



MIPS - ISA tip: LOAD / STORE:

- arhitectură cu 3 adrese
- registre de uz general
- procesor pe 32 biți.

MIPS – Registre CPU

- **32 x 32-biți Registre de uz general (General Purpose Registers - GPR)**
  - R0 – R31
  - R0 are valoarea fixă 0.
    - Tentativa de scriere în R0 este permisă, dar nu are efect
  - R31 memorează adresa de întoarcere la apelul de proceduri

- **PC – Program Counter / Contorul de Program**
  - Conține adresa instrucțiuni curente care se execută

- **Hi, Lo**
  - Stochează rezultatele parțiale pentru înmulțiri și împărțiri

MIPS – Registre CPU

Nume	Număr registru	Utilizare
$zero	0	Permanent 0
$at	1	Rezervat pentru asamblor
$v0 - $v1	2-3	Evaluarea expresiilor și rezultatul funcțiilor
$a0 - $a3	4-7	Parametri
$t0 - $t7	8-15	Valori temporare, salvate de apelant
$s0 - $s7	16-23	Valori temporare, salvate de funcția apelată
$t8 - $t9	24-25	Valori temporare, salvate de apelant
$k0 - $k1	26-27	Rezervați pentru nucleul OS
$gp	28	Pointer global
$sp	29	Pointer stivă (top of stack)
$fp	30	Pointer cadru (Frame Pointer, vârf stivă în apel)
$ra	31	Adresa de întoarcere, salvată de instrucțiunea de apel (call)

MIPS – registre GPR, convenții software

MIPS - Coprocesoare

Coprocesor 0 – CP0

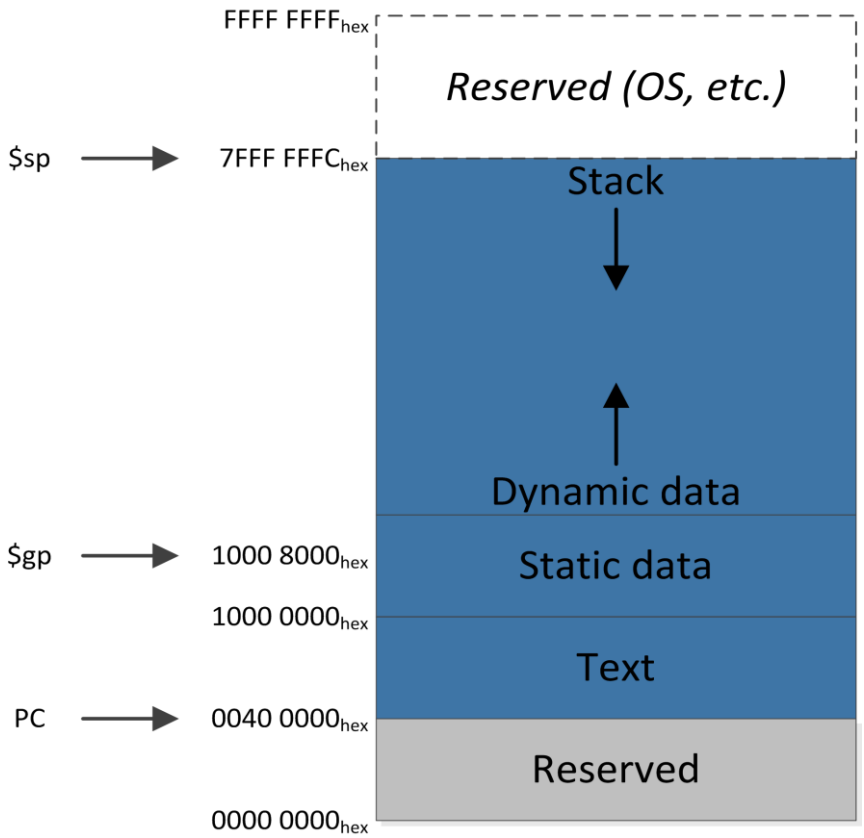
- Sprijină sistemul de operare: excepții, management memorie, etc.
- CP0 – mai mulți registre, dintre care
  - Registru de stare (CP0, Reg12) – nivelul de privilegii CPU, pini de întrerupere,
  - Registru de cauză (CP0, Reg13) – cauza pentru cea mai recentă întrerupere,
  - Registru EPC (CP0, Reg14) – PC – adresa de reluare a execuției după tratare,
  - Registru BadVAddr (CP0, Reg08) – adresa virtuală pentru cea mai recentă excepţie de adresă.
- Instrucțiuni pentru citirea și scrierea registrelor din Coprocesor 0
  - mfc0 $8, $13 # Move from Coprocessor 0, Reg13 (reg. de cauză) in $8
  - mtc0 $0, $12 # Move value 0 in Coprocessor 0, Reg12 (reg. de stare)

Coprocesor 1 – CP1; Floating Point Unit – Unitatea de virgulă mobilă

- 32 de registre (32 de biți) de virgulă mobilă – FPRs (f0 – f31);
- Cinci registre de comandă

Organizarea Memoriei

- Echivalentă cu un șir unidimensional, o adresă de memorie reprezintă o locație
- **"Byte addressing"**, se poate accesa fiecare octet
  - adresă pe 32-biți;
  - 2^{32} octeți cu adresele 0, 1, 2, to 2^{32}-1; ⇔ 2^{30} cuvinte / words
- Operaţii cu "words" → MIPS, un cuvânt - 32 biți sau 4 octeți
- Endianness se poate selecta, big/little endian pentru unele tipuri MIPS
- **Aliniere**
  - cuvânt 32-biți – adresă multiplă de 4 octeți
  - semi cuvânt 16-biți – adresă multiplă de 2 octeți

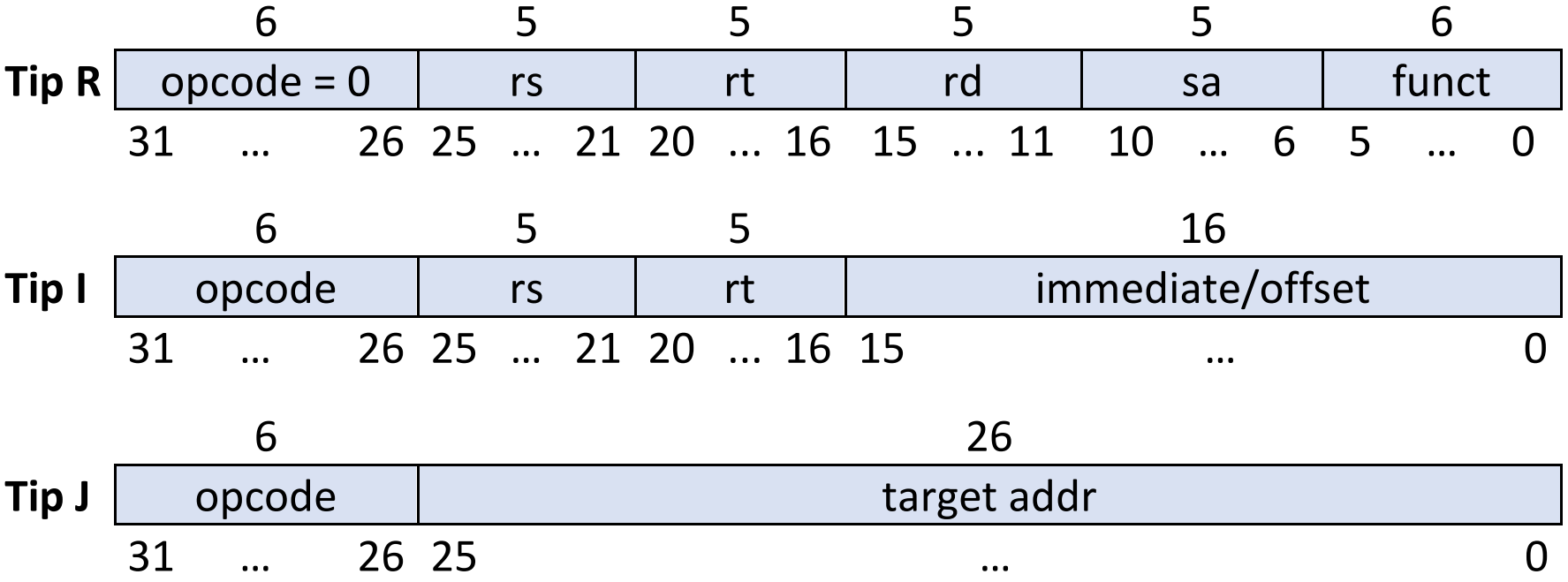


MIPS - Tipuri de date

- **Se operează pe:**
  - Întregi 32-bit (fără semn sau complement față de 2)
  - Virgulă mobilă 32-bit (precizie simplă), numere reale
  - Virgulă mobilă 64-bit (precizie dublă), numere reale
- **În registre de uz general GPR se pot încărca:**
  - Cuvinte 32-biți, semi-cuvinte 16-biți, și octeți, extinse la nevoie cu zero sau cu semn
- **În registre de virgulă mobilă FPR se pot încărca:**
  - Doar elemente de 32-biți
  - Numerele reale pe 32-biți pot fi stocate în oricare dintre registrele FPU
  - Numerele reale pe 64-biți pot fi stocate doar în registre FPU numerotate par (f0, f2, f4, etc)

MIPS - Formate de instrucțiuni

- Instrucțiuni de lungime 32-biți
- 3 tipuri/formate de instrucțiuni, 32 de biți
  - **Tip R** – folosite pentru operații aritmetice / logice (opcode = 0)
  - **Tip I** – folosite pentru operații aritmetice / logice cu valoare **I**mediată, lucru cu memoria, și ramificări condiționate
  - **Tip J** – folosite pentru salturi necondiționate

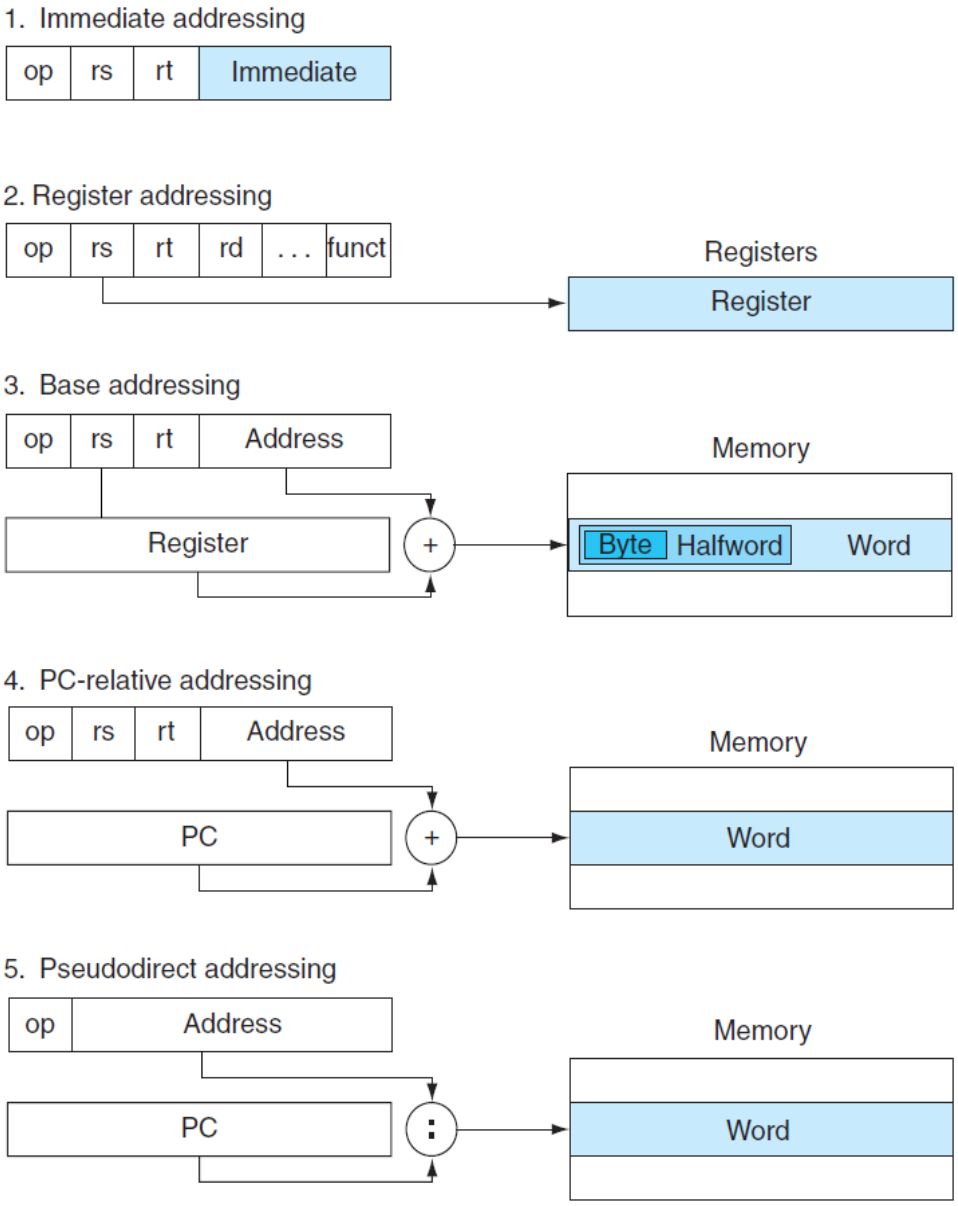


MIPS – tipuri/formate de instrucțiuni

MIPS - Formate de instrucțiuni

Câmp	Descriere
opcode	Codul de operație principal pe 6 biți
rs	Indexul primului registru sursă 5 biți
rt	Indexul celui de al doilea registru sursa / indexul registrului destinație / specifică funcția în cazul instrucțiunilor din clasa (cod operație) REGIMM, 5 biți
rd	Indexul registrului destinație 5 biți
sa	5 biți cantitatea de deplasare / shift amount
funct	6 biți câmpul de funcție care indică operația pentru codul de operație principal SPECIAL (ex. la tip R, opcode=0)
immediate / offset	Imediat pe 16 biți pentru: operanzi logici, operanzi aritmetici cu semn
	deplasamentul, în octeți, pentru operații cu memoria load/store
	deplasamentul cu semn, relativ la PC+4, în instrucțiuni
target address	index pe 26 biți care se deplasează cu 2 poziții la stânga pentru a calcula cei mai puțin semnificativi 28 de biți ai adresei de salt

MIPS - Moduri de adresare



1. **Adresare imediată**
2. **Adresare cu registre**
3. **Adresare cu bază (indexată) a memoriei**
   - *indexat*; conținut rs + Address,
   - Address= 0 → *register indirect*
   - dacă rs=r_0: r0 + Address → *adresare absolută*
4. **Ramificări – salt condiționat** – adresare relativă la PC:
   - adresa ramificare = PC + 4 + 4×Address
5. **Salt**: Adresare Pseudo-directă – noua adresă este PC(31:28) || 28-biți Address; salt în regiuni de 256 MB

MIPS – Instrucțiuni de bază (pe categorii)

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 – Instrucțiuni

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 – Instrucțiuni

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, $rs, sa	R	RF[rd] ← RF[rs] << sa	Shift Left Logical
srl $rd, $rs, sa	R	RF[rd] ← RF[rs] >> sa	Shift Right Logical
sra $rd, $rs, sa	R	RF[rd] ← RF[rs] >> 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 – Instrucțiuni

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 $rs, $rt, imm	I	If(RF[rs] == RF[rt]) then PC ← PC + 4 + S_Ext(imm) <<2	Branch if equal
bne $rs, $rt, imm	I	If(RF[rs]!=RF[rt]) then PC ← PC + 4 + S_Ext(imm) <<2	Branch if 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 – Instrucțiuni

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] < 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] < 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 – Instrucțiuni

- **Instrucțiuni aritmetice / logice tip R**

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

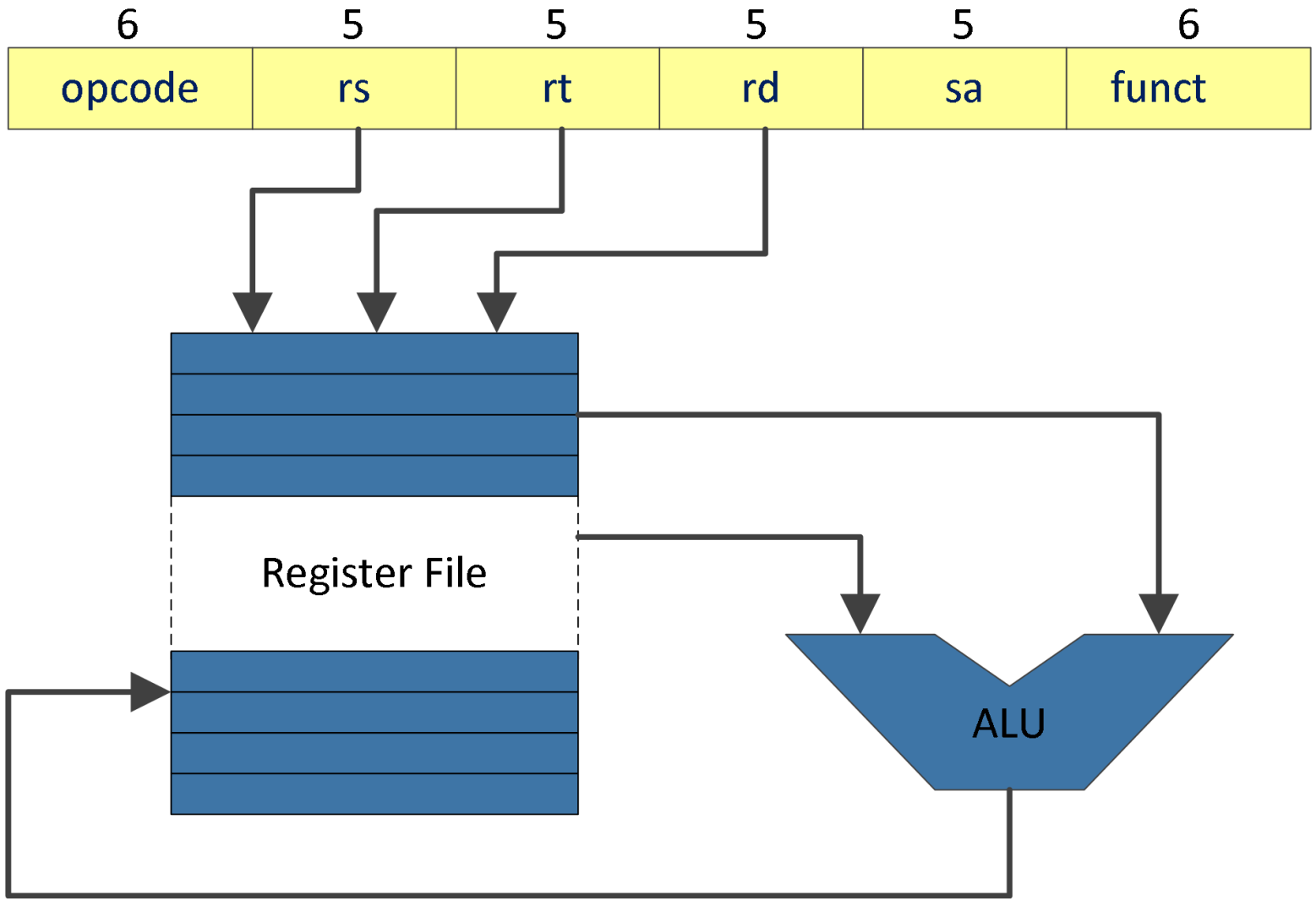
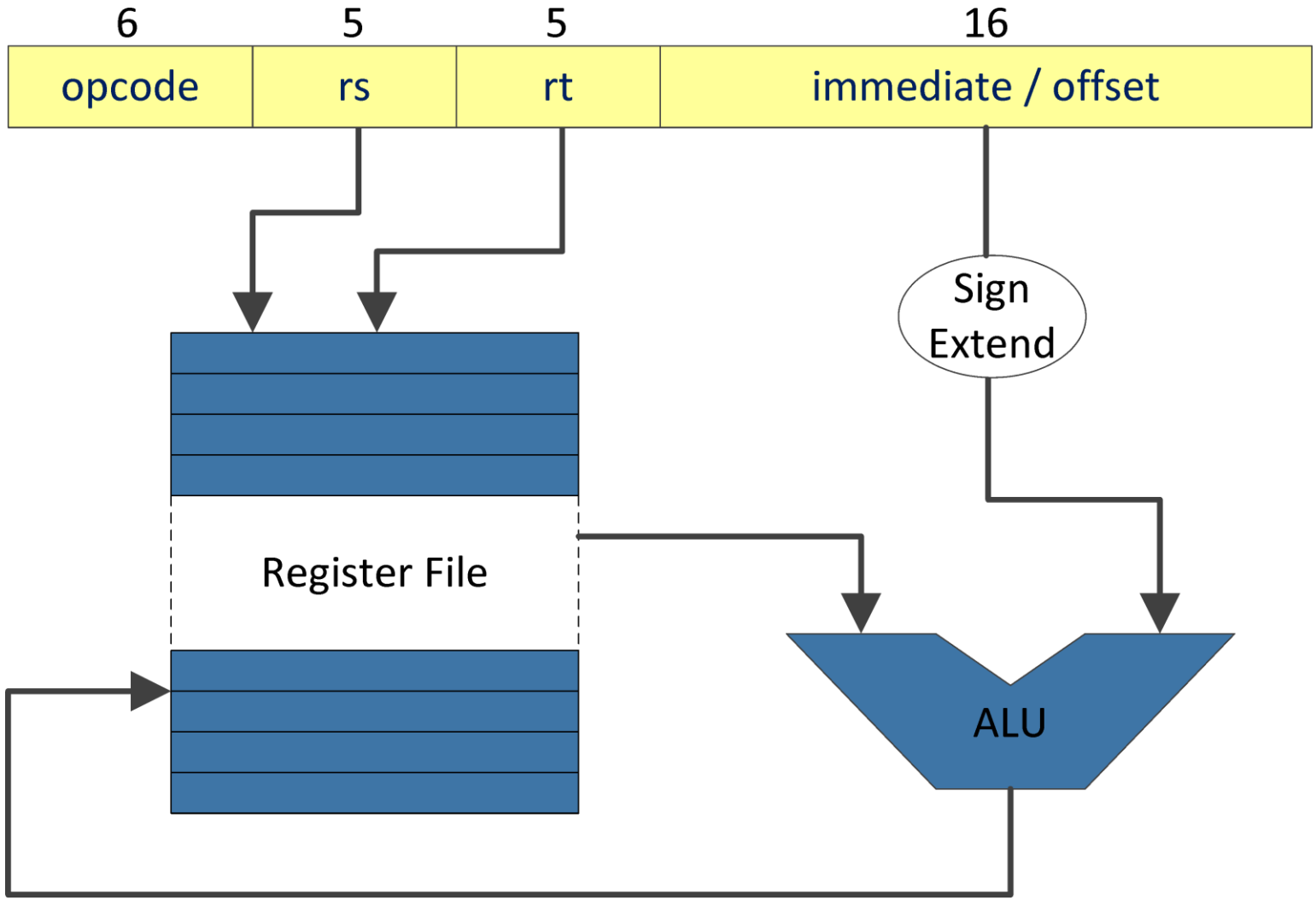


Diagrama de procesare (formă grafică pentru transferul între registre)

MIPS – Instrucțiuni

- **Instrucțiuni aritmetice cu imediat, tip I**

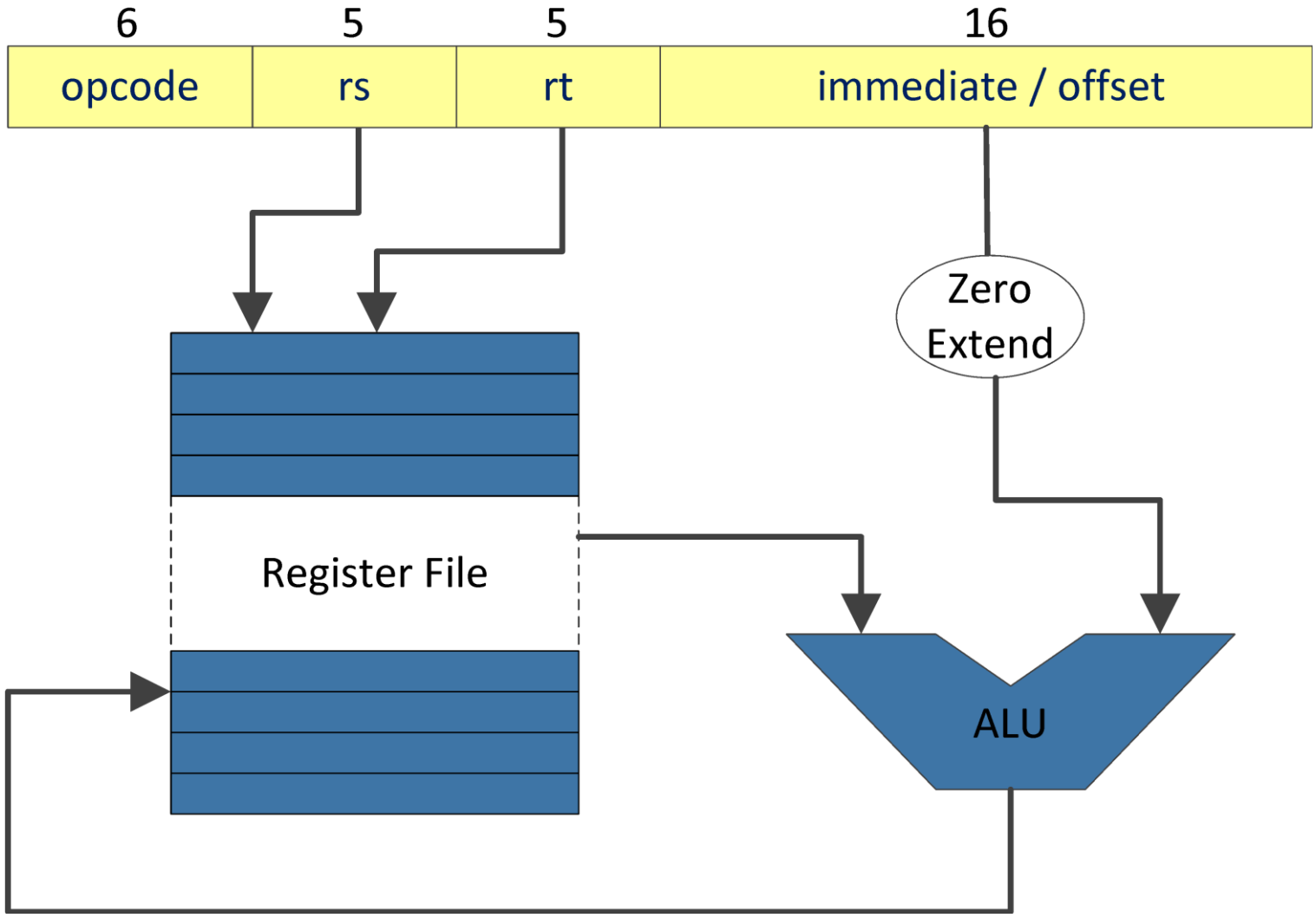
addi $rt, $rs, imm → RF[rt] ← RF[rs] + S_Ext(imm)



MIPS – Instrucțiuni

- **Instrucțiuni logice cu imediat, tip I**

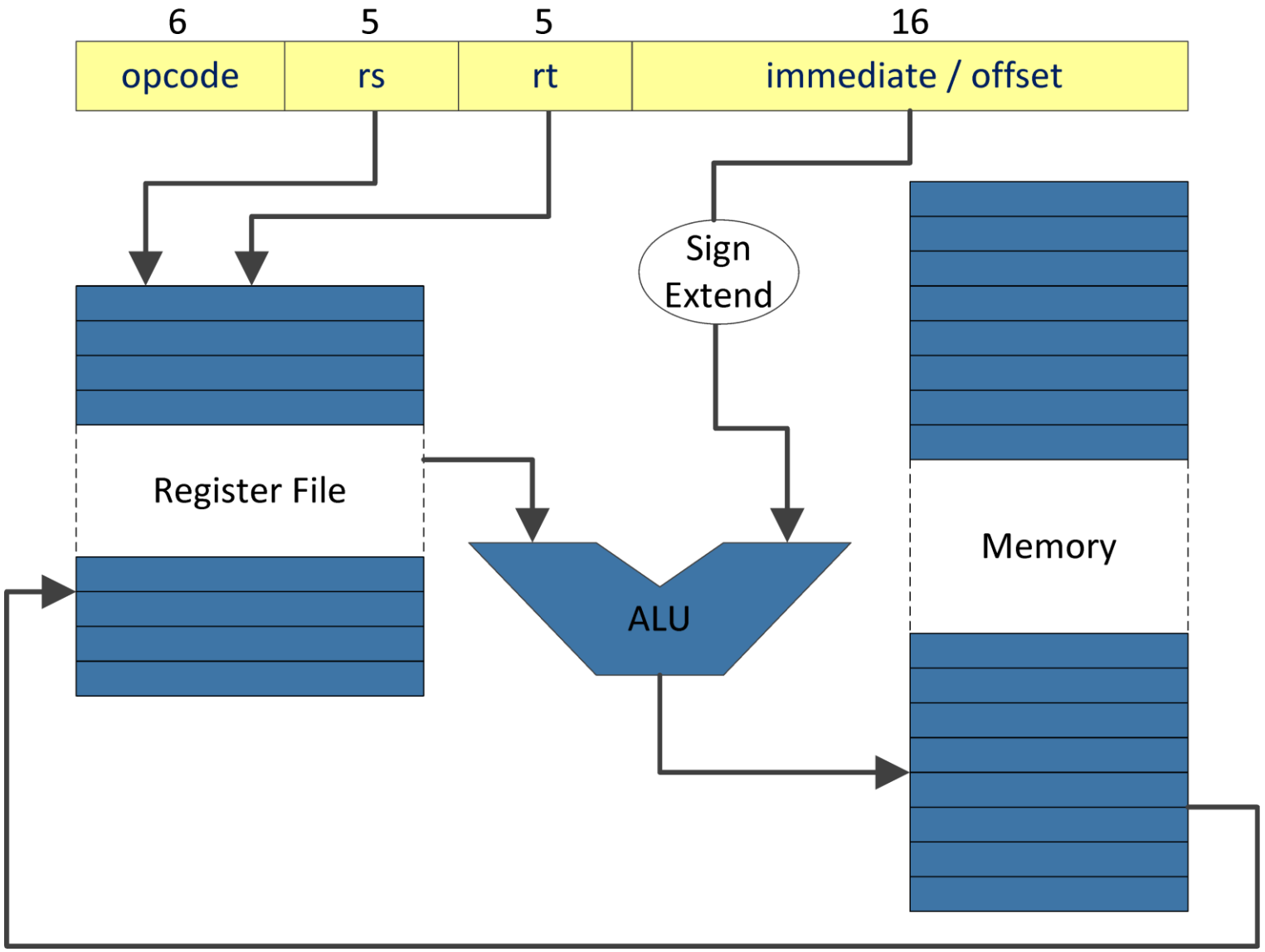
ori $rt, $rs, imm → RF[rt] ← RF[rs] or Z_Ext(imm)



MIPS – Instrucțiuni

- **Instrucțiunea Load Word, tip I**

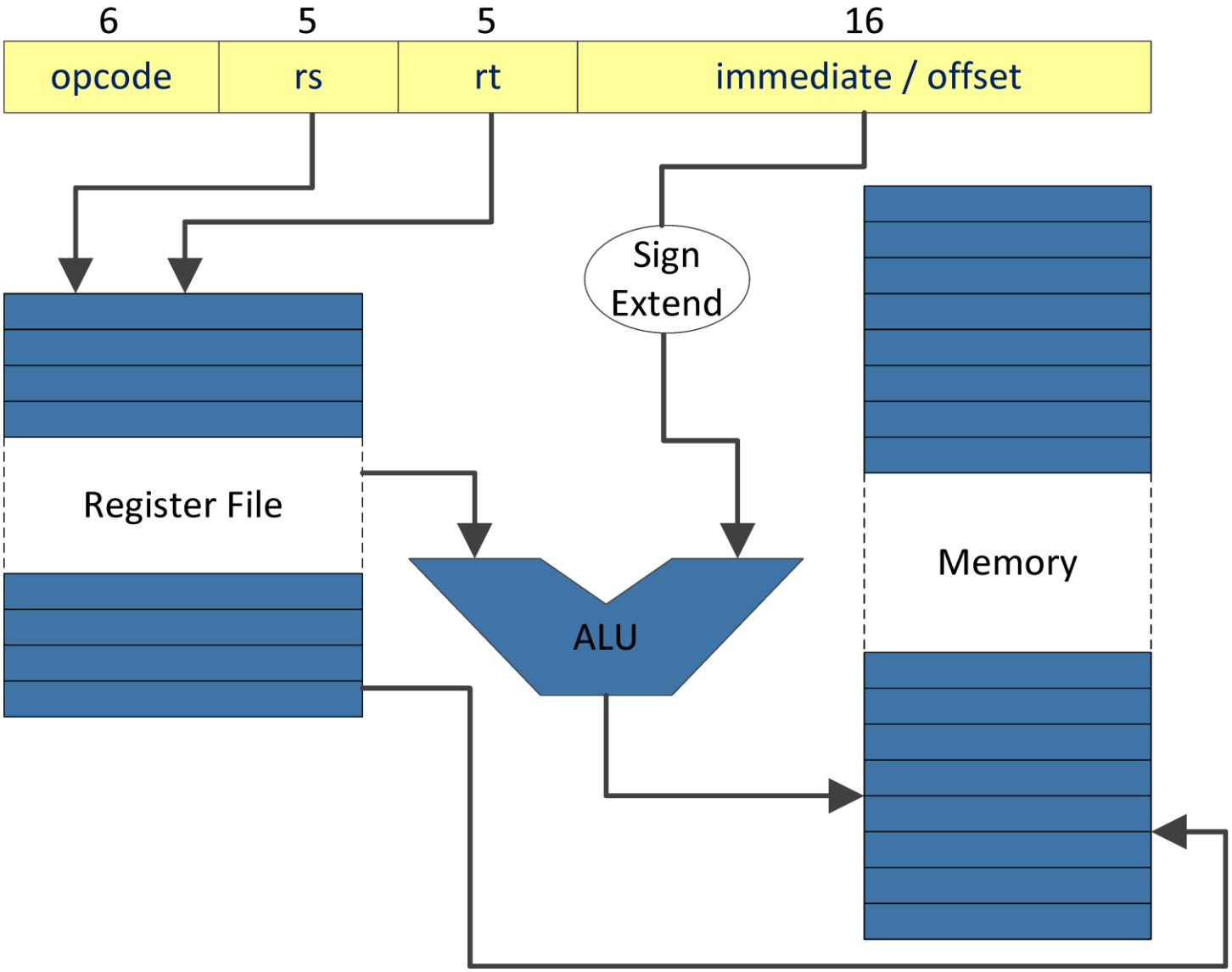
lw $rt, imm($rs) → RF[rt] ← M[RF[rs] + S_Ext(imm)]



MIPS – Instrucțiuni

- **Instrucțiunea Store Word, tip I**

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

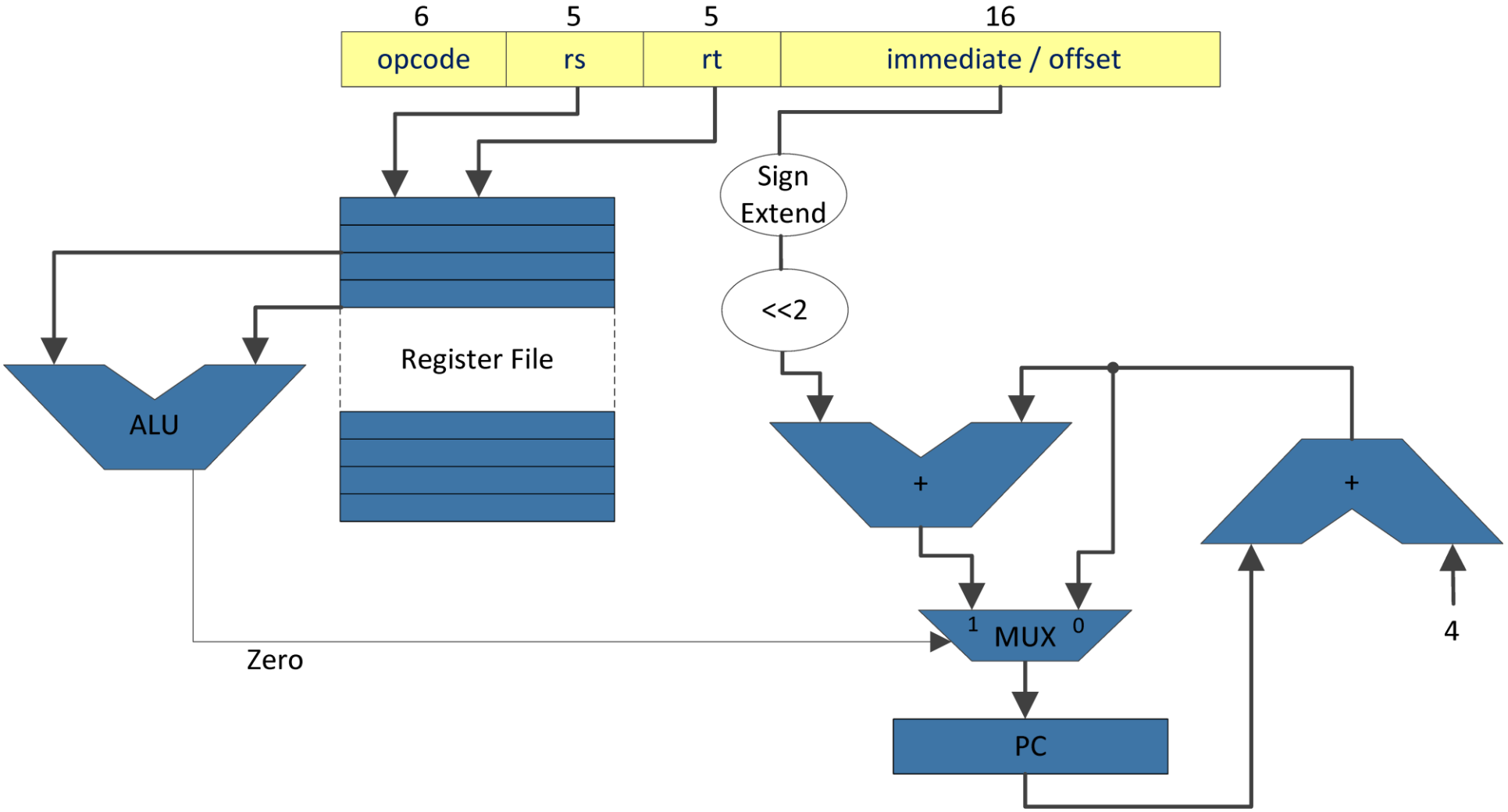


MIPS – Instrucțiuni

- **Instrucțiunea Branch If Equal, tip I**

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

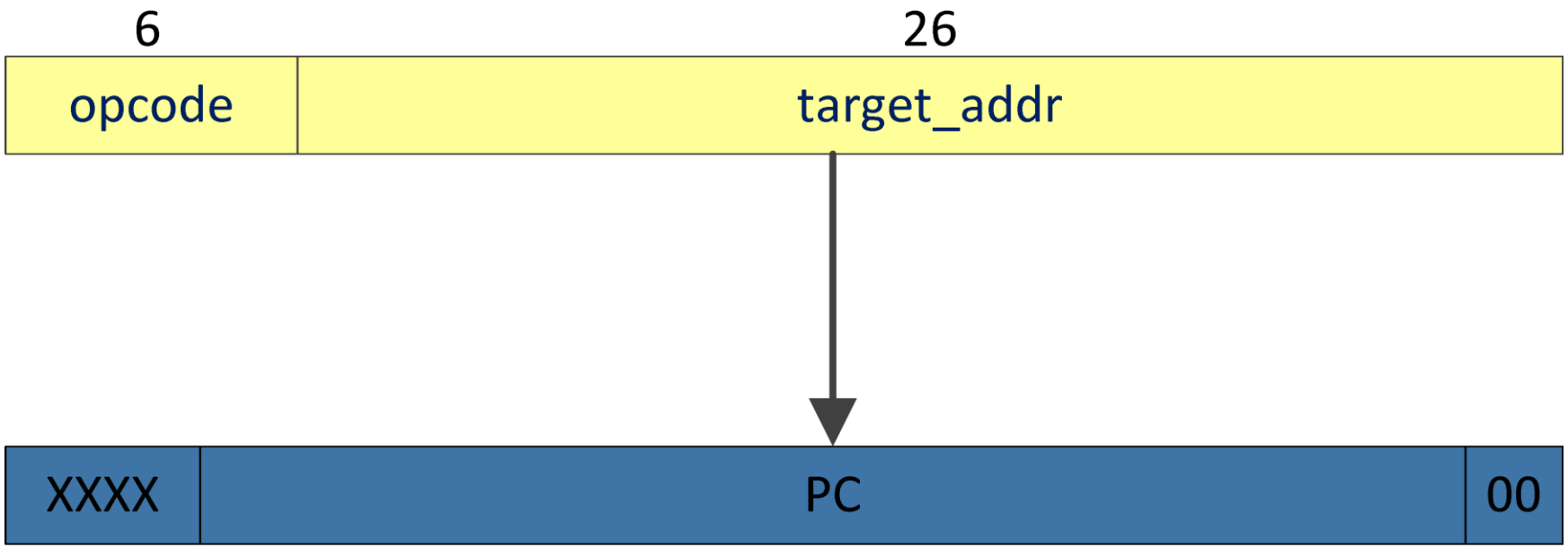
else PC ← PC+4



MIPS – Instrucțiuni

- **Instrucțiunea Jump, tip J**

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



MIPS - Pseudo instrucțiuni

- Sunt instrucțiuni care **nu sunt suportate** de procesor, dar **pot fi suportate de către asamblor**, fiind implementate **prin instrucțiuni native** ale procesorului
- Sunt utile pentru programator pentru a implementa operații uzuale care nu sunt suportate nativ ca instrucțiuni de către procesorul MIPS

Ex.:

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

MIPS – tratarea întreruperilor

- MIPS hardware pentru tratarea întreruperilor → 3 pași, este implicat coprocesorul 0

Pasul 1. Se salvează conținutul PC
- Registrul EPC (Exception Program Counter), memorează în funcție de eveniment:
  - Adresa instrucțiunii care generează o excepție (ex.. eroare de adresare, instrucțiune rezervată) sau eroare hardware (ex. memory parity error)
  - Adresa instrucțiunii următoare, ex. întreruperi externe

Pasul 2. PC – salt la rutina de tratare a întreruperilor și se salvează Cauza
- PC ← 80000180_{16}, adresa depinde de tipul MIPS
- Cause register ← cod cauză întrerupere
- Fiecare tip de întrerupere are un cod, ex.:
  - Întrerupere hardware = 0
  - Adresa memorie ilegală (load/fetch sau store) = 4 sau 5
  - Eroare magistrală (fetch sau load/store)= 6 sau 7
  - Instrucțiune syscall = 8
  - Cod operație ilegal, rezervat sau nedefinit = 10
  - overflow la operații pe întreg = 12
  - orice excepție virgula flotantă = 15

Pasul 3. Mod funcționare CPU devine - kernel mode (supervizor)
- CPU mode bit ← 0;

Probleme / Temă

1. Care este spațiul de adresare pentru procesorul MIPS (memoria care poate fi accesată) ?
2. Folosind instrucțiuni MIPS native, scrieți pseudo instrucțiuni pentru următoarele operații:
   - Valoare absolută (modul)
   - Branch on Greater Than
   - Branch on Less or Equal
   - Swap two registers (schimbă două registre)

- Definiți instrucțiuni noi pentru:
  - LWR (load word register) – însumează două registre pentru calculului adresei de memorie de unde să citească.
  - SWR (store word register) – idem, pentru memorare în memorie.
  - LWA – folosește un singur registru pentru calculul adresei.
  - SWA – idem.
  - Pași: stabiliți formatul de instrucțiune optim, RTL abstract, și desenați diagrama de procesare.

Referințe

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”.