Organisasi dan Arsitektur Komputer (COA)
CII2A3

Instruksi MIPS

Disusun oleh: Tim dosen COA

Fakultas Informatika
Universitas Telkom

Desember 2022

Rencana Studi

Fakultas Informatika
School of Computing
Telkom University

Rencana Studi		Rincian Nilai Kuis																Rincian Nilai Tugas									Rincian Nilai Hasil Proyek											Bobot Tiap CLO (%)			
Pertemuan Ke-	Materi	CLO 1			CLO 2					CLO 3				CLO 4				CLO 1			CLO 2					CLO	CLO 3						CLO 4										
		Kuis (Kognitif) (20 %)																Tugas Partisipatif (35%)									Hasil Proyek (45%)															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1a	1b	1c	2a	2b	2c	2d	2e	3a	3b	1a	1b	2a	1c	2b	2c	3a	3b	4a	3c	4b	3d	4c	1	2	3	4
1	Sistem komputer	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
2	Input/Output	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	14	--	--	--
3	Sistem Bus	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
4	Organisasi memori	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
5	Cara kerja memori utama (RAM)	-	-	-		1	-	-	-	-	-	-	-	-	-	-		-	-	-		4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
6	Memori sekunder	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	--	26	--	--
7	Cara kerja *cache memory* (bag-1)	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
8	Cara kerja cache memory (bag-2)	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
9	Arsitektur SAP-1	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-				
10	Arsitektur SAP-2	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	2	1	-	-	-	-	-	-	-	-	-	-	--	--	30	--
11	Arsitektur SAP-3	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-	-	-	-	7	2		-	-	-	-	-	-	-				
12	Instruksi *Extended* dan *Indirect*	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7	-	-	-	-	-	-	-	-				
13	Arsitektur MIPS	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-				
14	Instruksi MIPS	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	1	-	-	-	-	-	--	--	--	30
15	Assembly MIPS (bag-1)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	2	-	-	-				
16	Assembly MIPS (bag-2)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9	7	-				

Eksekusi Instruksi

Instruksi

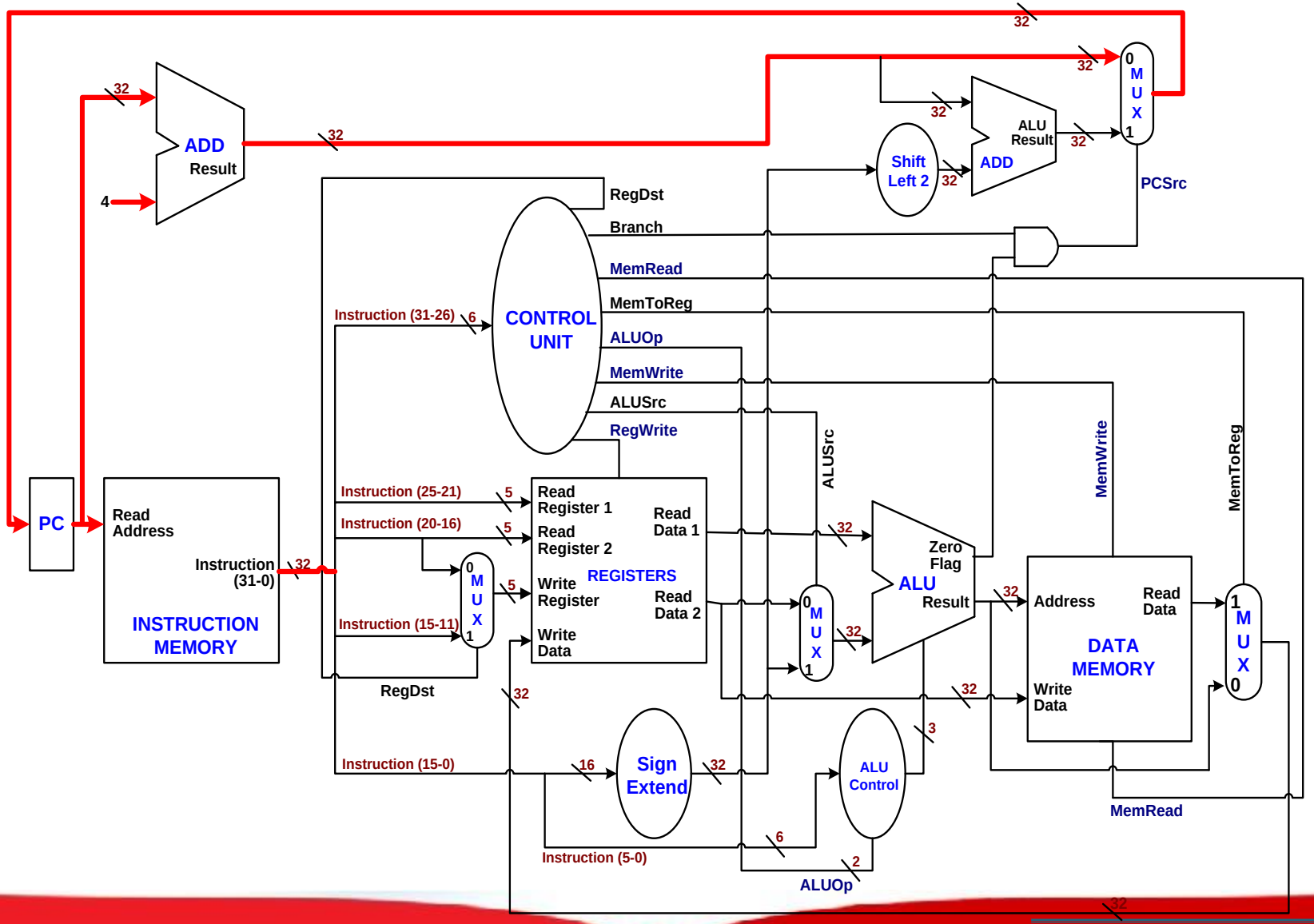
- Instruksi disimpan dalam memori pada alamat awal
- Data disimpan di memori pada alamat akhir
- Bagaimana mengeksekusi instruksi?
- Pada umumnya terdapat tiga tahap:
  - *Fetch*: Pengambilan
  - *Decode*: Penerjemahan
  - *Execute*: Eksekusi
- Pada beberapa prosesor tahapannya bisa lebih dari 3 tahap

Fetch (1)

- Tujuan: untuk mengambil instruksi dari memori instruksi ke register
- Instruksi memiliki lebar 32 bit
- Setiap 8 bit menggunakan 1 alamat memori, sehingga satu instruksi menempati 4 alamat
- Karena satu instruksi disimpan dalam 4 alamat maka PC akan ditambah nilainya dengan 4 setiap kali selesai mengambil instruksi

Fetch (2)

Fakultas Informatika
School of Computing
Telkom University

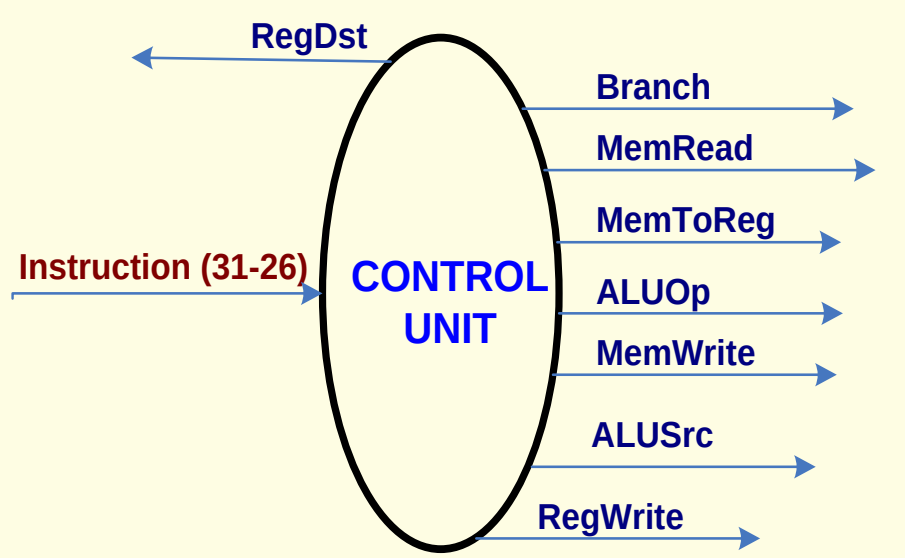


Fetch (3)

- Misal: instruksi add $t2, $s2, $t1
  - Bahasa mesin:

    00000010 01001001 01010000 00100000
- Disimpan dalam memori:
  - Address 0 00000010
  - Address 1 01001001
  - Address 2 01010000
  - Address 3 00100000
- Fetch:
  - Instruction Register ← 00000010010010010101000000100000
  - PC ← PC+4 # untuk mengambil instruksi berikutnya

Decode (1)



- Tujuan: untuk menterjemahkan kode operasi (instruction[31-26]) dan menghasilkan 9 bit kontrol (microinstruction)
- Opcode: 6 bits awal instruksi yang terdapat dalam register instruksi
- Unit kendali mendefinisikan tipe instruksi dan membuat 9 bit kendali

Decode (2)

- Dalam unit kendali terdapat ROM yang berguna untuk menterjemahkan instruksi
- Kode operasi sebagai input akan diproses sehingga dikeluarkan satu kombinasi jalur kendali 9 bit
- Nilai yang keluar dari unit kendali disebut mikroinstruksi
- Mikroinstruksi mengendalikan operasi pada *datapath*

Execute

- Tujuan: mengeksekusi instruksi
- Instruksi dibagi menjadi tiga kelas:
  - R-type
  - Load/Store-type
  - Branch-type
- Tipe R adalah tipe aritmetik
- Tipe Load/Store melakukan pengaksesan memori baik menyimpan maupun membaca
- Tipe Branch digunakan untuk instruksi lompatan dan pencabangan
- Setting jalur kendali tergantung field opcode pada instruksi:

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-Format	1	0	0	1	0	0	0	1	0
Lw	0	1	1	1	1	0	0	0	0
Sw	X	1	X	0	0	1	0	0	0
Beq	X	0	X	0	0	0	1	0	1

Jenis Instruksi MIPS

- Instruksi pada MIPS dibagi dalam tiga bentuk format yakni: format R, format I, format J
  - R format terdiri dari tiga register dan *function field*
  - I format terdiri atas dua register dan 16 bit *long immediate value*
  - J format terdiri atas enam bit opcode yang diikuti oleh 26 bits *immediate value*

Instruksi Tipe-R (1)

0	rs	rt	rd	shamt	fuct
31-26	25-21	20-16	15-11	10-6	5-0

- **0:** opcode (6 bit = 000000)
- **rs:** register source (5 bit) = Read register 1 = nomor register yang digunakan untuk menaruh operand pertama
- **rt:** register target (5 bit) = Read register 2 = nomor register yang digunakan untuk menaruh operand kedua
- **rd:** register destination (5 bit) = Write register = nomor register yang digunakan untuk menaruh hasil dari ALU
- **shamt:** tidak digunakan (5 bit = 00000)
- **funct:** jenis fungsi yang akan dilakukan oleh ALU (6 bit) = and, or, add, substract, dan set on less than

Instruksi Tipe-R (2)

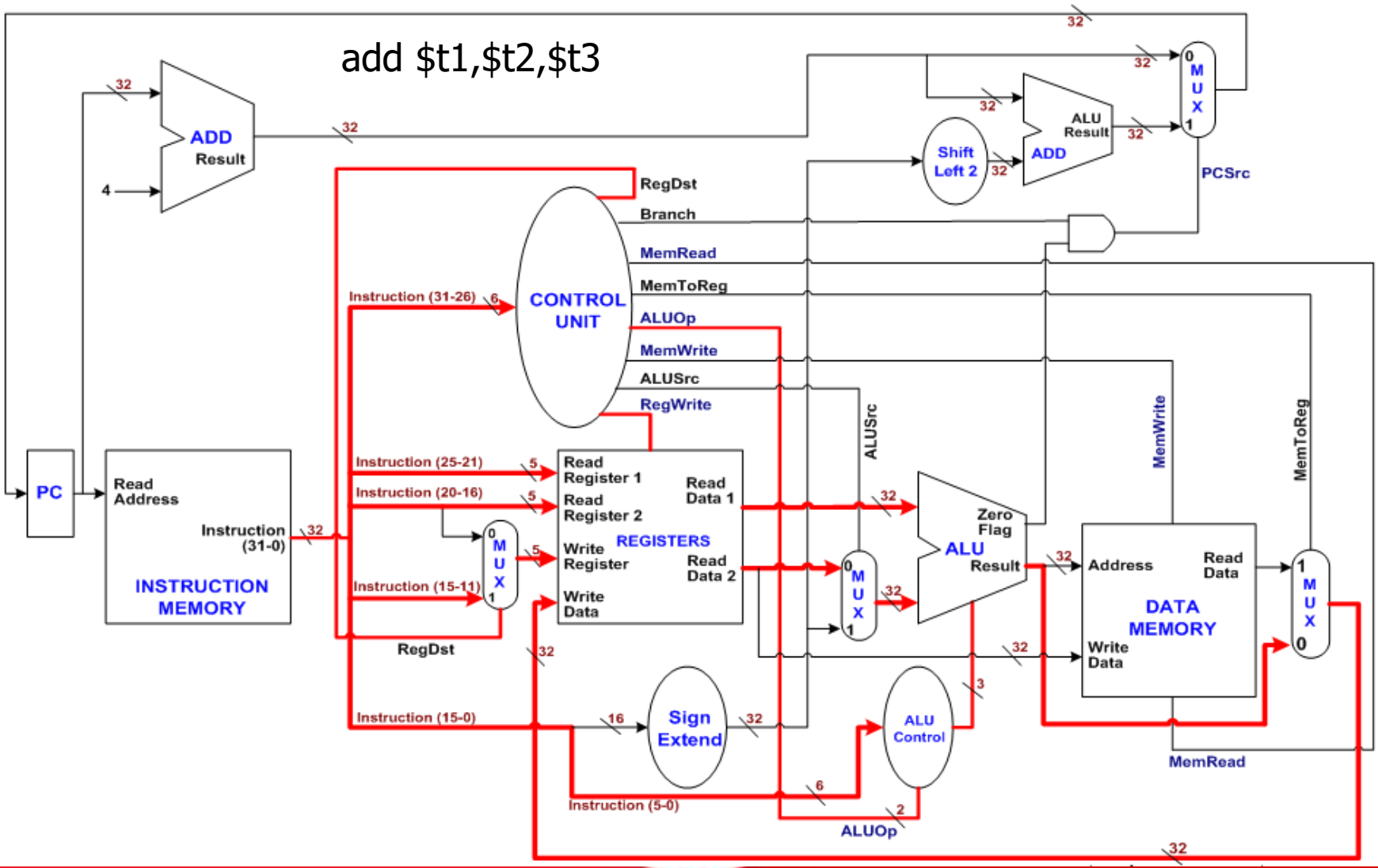
- Pada instruksi tipe R kode operasinya sama yaitu 000000
- Pembeda satu instruksi dengan instruksi yang lain adalah pada 6 bit bagian bawah (LSB) sebagai fungsi operasi aritmetik
- Untuk melakukan operasi ALU sesuai dengan instruksi digunakan control ALU dengan *input* 6 bit terbawah dari instruksi

Instruksi Tipe-R (3)

- Terdapat beberapa komponen yang terlibat pada pengeksekusian tipe aritmatik:
  - Register
  - ALU
  - MUX
  - ALU control
- Keempat komponen tersebut beroperasi saling bebas

Instruksi Tipe-R (4)

Instruction	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-Format	1	0	0	1	0	0	0	1	0



Instruksi Tipe-R (5)

Contoh operasi R-Type: add $t1,$t2,$t3

1. Ambil instruksi dari memori instruksi pada alamat yang terdapat di dalamPC
2. Update isi PC dengan menambah 4
3. Baca isi dua register yaitu $t2 (Read register 1) dan $t3 (Read register 2) dari register file
4. Unit kendali menentukan setting baris kendali
5. ALU mengoperasikan data yang dibaca dari register file menggunakan kode fungsi (yaitu bit 5 – 0 atau field funct dari instruksi) untuk menghasilkan fungsi ALU
6. Hasil dari ALU dituliskan ke dalam register file menggunakan bit 15-11 dari instruksi untuk memilih register tujuan $t1 (Write register)

Instruksi Tipe-R (6)

- Nilai tiga bit jalur kendali ALU

ALU control lines	Function
000	and
001	or
010	add
110	subtract
111	set on less than

- Input kendali ALU berasal dari 6 bit terbawah instruksi
- Kendali ALU dikendalikan oleh ALUOp

Instruksi Tipe-R (7)

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	Load word	XXXXXX	Add	010
SW	00	Store word	XXXXXX	Add	010
Branch equal	01	Branch equal	XXXXXX	Substract	110
R-type	10	Add	100000	Add	010
R-type	10	Substract	100010	Substract	110
R-type	10	AND	100100	And	000
R-type	10	OR	100101	Or	001
R-type	10	Set on less than	101010	Set on less than	111

- Kendali operasi pada ALU ditentukan oleh ALUOp dan *field* fungsi
- Bagaimana bit kendali ALU diset tergantung dari bit kendali ALUOp dan fungsi yang berbeda untuk instruksi tipe-R

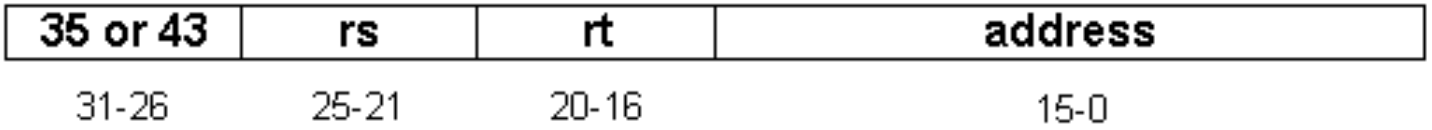
Instruksi Tipe-R (8)

- Tabel kebenaran untuk tiga bit kendali ALU

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

- Pada saat ALUOp bernilai 00 (LW dan SW) atau X1 (*branch*), maka field fungsi tidak digunakan
- Pada saat ALUOp bernilai 1X (R-type), maka *field* fungsi digunakan untuk menentukan operasi yang akan dilakukan oleh ALU

Instruksi Tipe *Load* atau *Store*

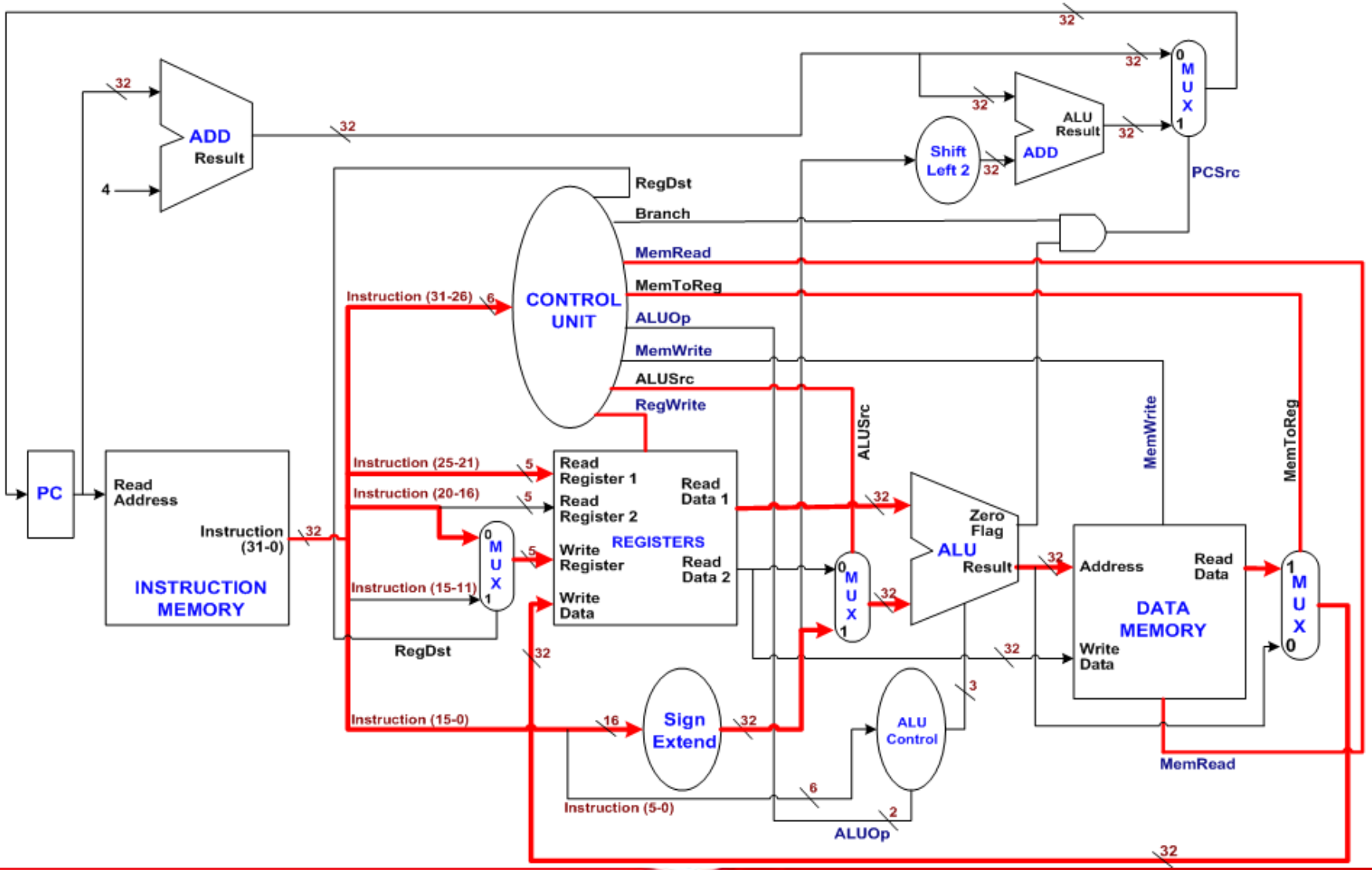


- 35 or 43: opcode (6 bit)
  - 35 = load = 100011
  - 43 = store = 101011
- rs: register source (5 bit)
  - Operasi **Load**: rs = Read register 1 = nomor register yang menyimpan **alamat dasar** *(base address)* dari data yang akan dibaca
  - Operasi **Store**: rs = Read register 1 = nomor register yang menyimpan **alamat dasar** *(base address)* dari memori dimana data akan disimpan
- rt: register target (5 bit)
  - Operasi **Load**: rt = Write register = nomor register yang akan digunakan untuk **menampung data** yang akan dibaca
  - Operasi **Store**: rt = Read register 2 = nomor register yang **menampung data** yang akan ditulis ke memori
- address (16 bit): *offset* (banyaknya pergeseran alamat dari alamat dasar)
- Contoh instruksi:
  - lw rt, physical address → lw $t1, offset($t2)
    - Alamat fisik = alamat dasar *(base)* + *offset*
      = isi $t2 + offset
  - sw rt, physical address → sw $t1, offset($t2)

Instruksi Tipe *Load* (1)

Fakultas Informatika
School of Computing
Telkom University

Instruction	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
Lw	0	1	1	1	1	0	0	0	0



Instruksi Tipe *Load* (2)

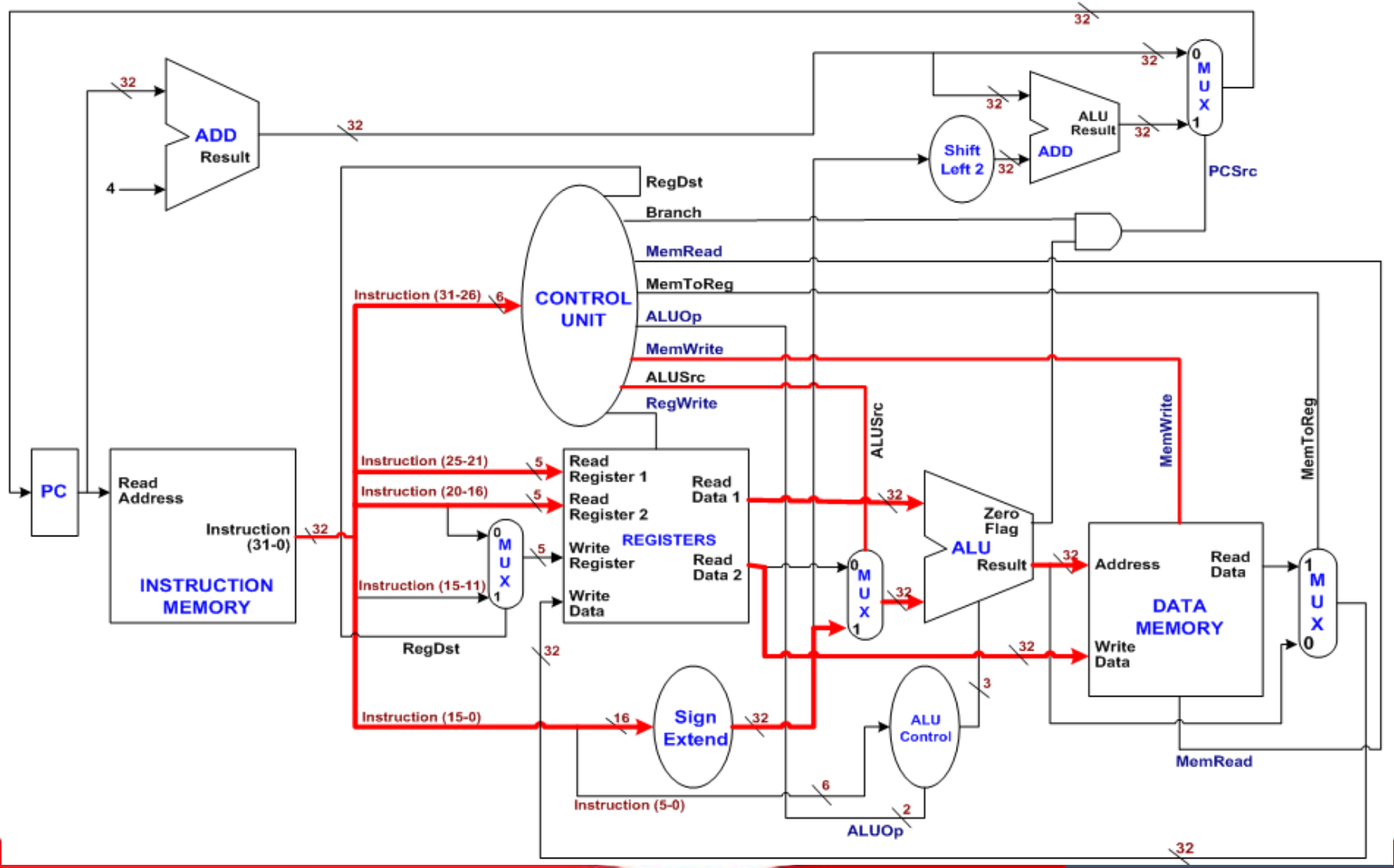
Contoh operasi tipe Load: lw $t1, offset($t2)

1. Ambil instruksi dari memori instruksi pada alamat yang terdapat di dalam PC
2. Update isi PC dengan menambah 4
3. Baca nilai register $t2 (*Read register 1*) dari register file. Nilai register $t2 merupakan alamat dasar (*base*) dari data yang akan dibaca
4. Nilai *offset* diubah menjadi 32 bit oleh *sign-extend*
5. ALU menghitung alamat data yang akan dibaca dengan cara menjumlahkan nilai yang dibaca dari register file dan *sign-extended*
6. Hasil penjumlahan digunakan sebagai alamat untuk membaca data dari memori data
7. Data dari memori dituliskan ke dalam register file. Register tujuan ditunjukkan oleh bit instruksi 20-16 atau $t1 (*Write register*)

Instruksi Tipe *Store* (1)

Fakultas Informatika
School of Computing
Telkom University

Instruction	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
Sw	X	1	X	0	0	1	0	0	0



Instruksi Tipe *Store* (2)

Contoh operasi tipe Store: sw $t1, offset($t2)

1. Ambil instruksi dari memori instruksi pada alamat yang terdapat di dalam PC
2. Update isi PC dengan menambah 4
3. Baca nilai register $t2 (*Read register 1*) dari *register file*. Nilai register $t2 merupakan alamat dasar (*base*) dari memori yang akan digunakan untuk menyimpan data
4. Nilai offset diubah menjadi 32 bit oleh *sign-extend*
5. ALU menjumlahkan nilai yang dibaca dari *register file* dan *sign-extended*
6. Hasil penjumlahan digunakan sebagai alamat memori yang akan digunakan untuk menaruh data
7. Data dari register (*Data read 2*) dituliskan ke dalam memori

Instruksi Tipe *Branch* (1)

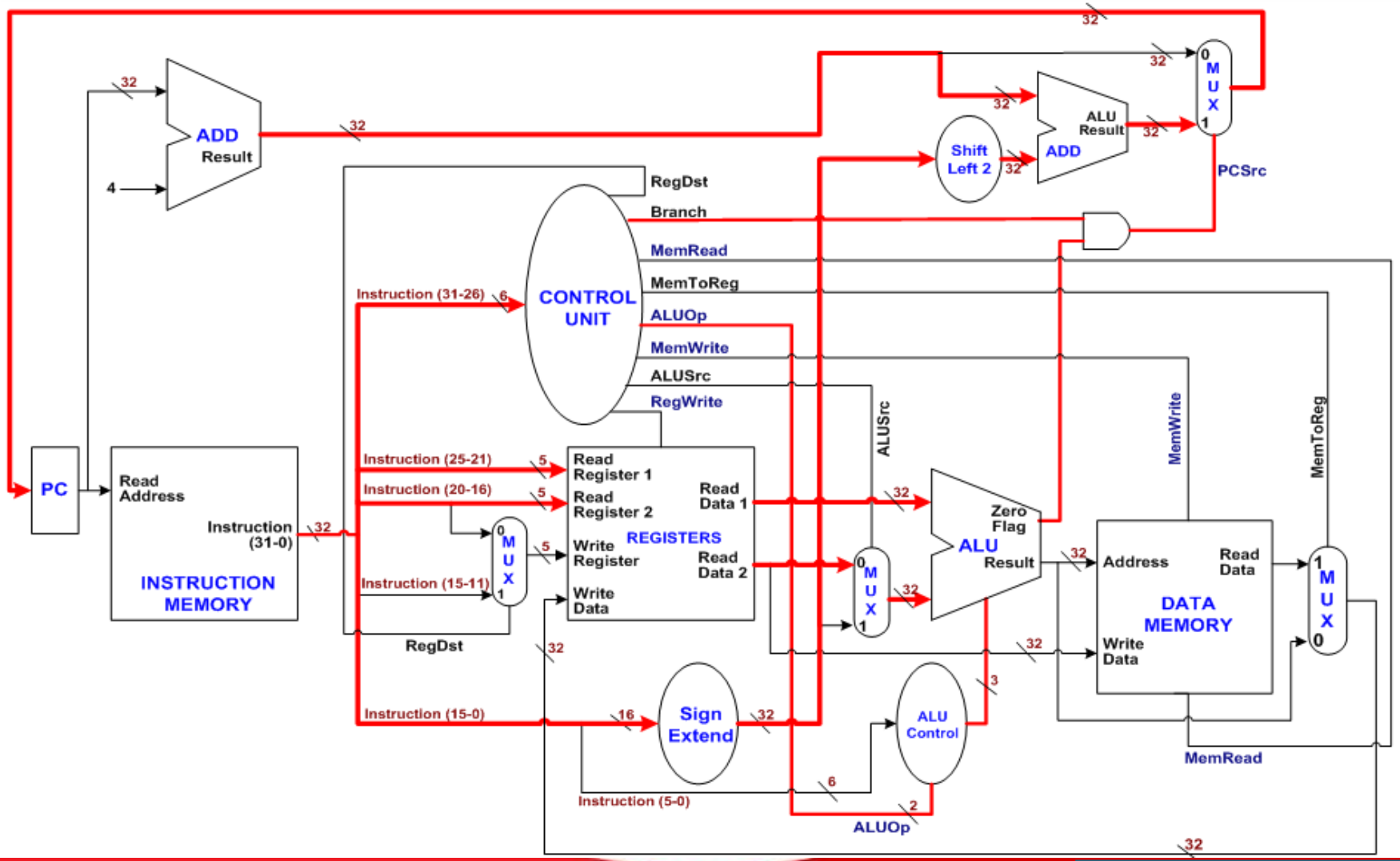
4	rs	rt	address
31-26	25-21	20-16	15-0

- **4:** *opcode* (6 bit = 000100)
- **rs:** *register source* (5 bit) = Read register 1 = nomor register yang menyimpan *operand* pertama yang akan dibandingkan
- **rt:** *register target* (5 bit) = Read register 2 = nomor register yang menyimpan operand kedua yang akan dibandingkan
- **address:** *offset* alamat memori (16 bit)
- Contoh instruksi:
  - Beq rs, rt, label → beq $t1, $t2, offset

Instruksi Tipe *Branch* (2)

Fakultas Informatika
School of Computing
Telkom University

Instruction	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
Beq	X	0	X	0	0	0	1	0	1



Instruksi Tipe *Branch* (3)

Contoh operasi tipe Branch: beq $t1,$t2,offset

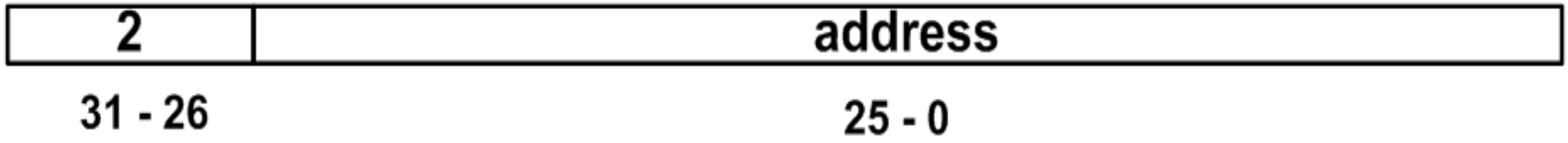
1. Ambil instruksi dari memori instruksi pada alamat yang terdapat di dalam PC
2. Update isi PC dengan menambah 4
3. Baca isi register $t1 dan $t2 (*Read register 1* dan *Read register 2*) dari *register file*
4. ALU melakukan pengurangan isi register $t1 dengan isi register $t2.
5. Zero flag di-set 1 jika hasilnya nol
6. Nilai *offset* diubah menjadi 32 bit oleh *sign-extend*
7. Nilai sign-extended digeser ke kiri 2 bit kemudian ditambahkan dengan PC+4 untuk memperoleh alamat tujuan pencabangan
8. Zero flag pada ALU akan menentukan terjadi pencabangan atau tidak (menentukan nilai PC selanjutnya)
   - Jika terjadi pencabangan:
     PC = Shift_Left_2_bit (sign-extended) + (PC+ 4)
   - Jika **tidak** terjadi pencabangan:
     PC = PC+ 4

Fungsi Kendali

- Fungsi kendali untuk implementasi satu siklus ditunjukkan pada tabel kebenaran di bawah ini

		= 0	= 35	= 43	= 4
Input or output	**Signal name**	**R-format**	**lw**	**sw**	**beq**
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

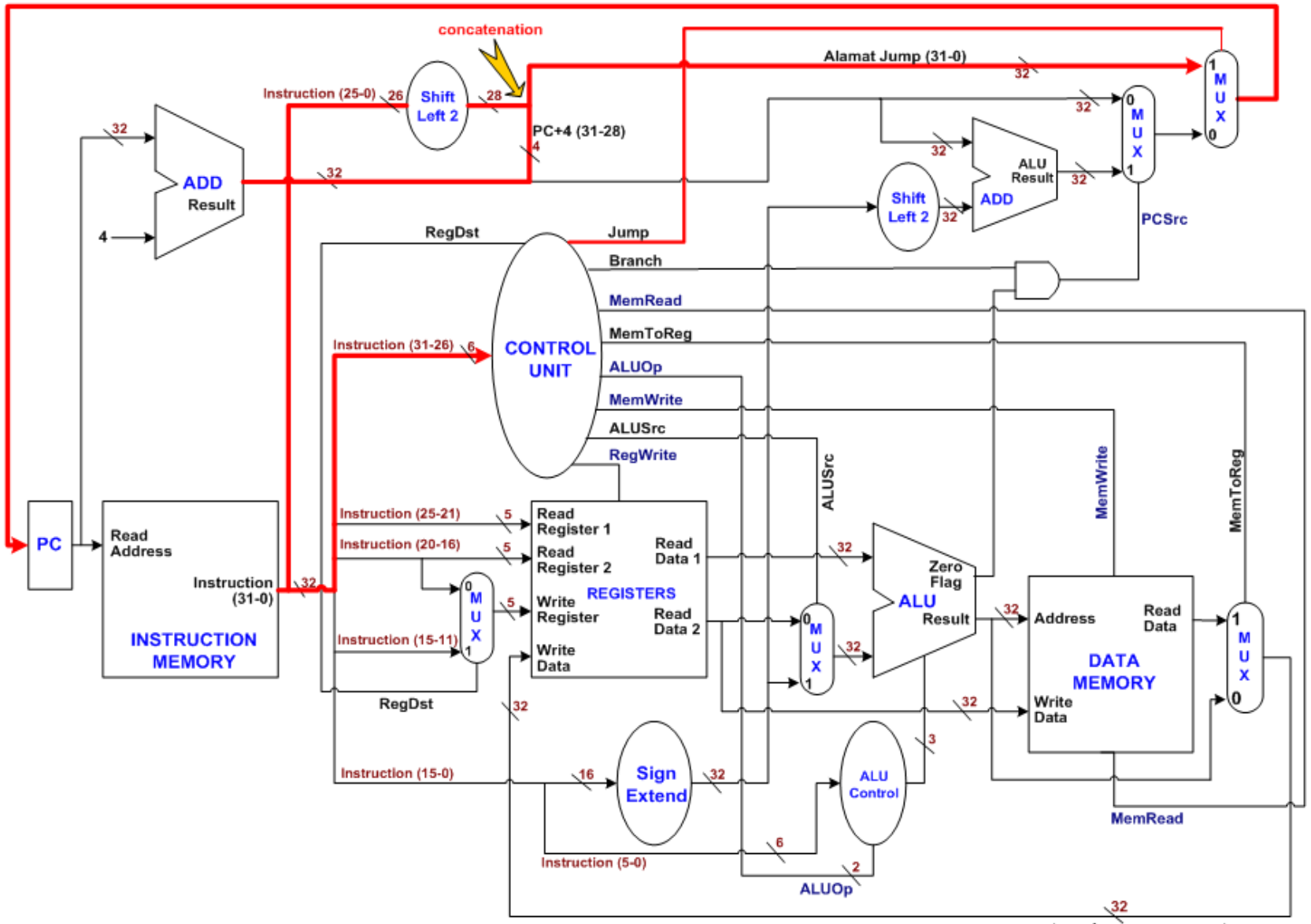
Instruksi Jump (1)



- **2:** opcode (6 bit = 000010)
- **address:** alamat untuk *jump* (26 bit) = alamat *immediate*
- Instruksi *Jump* mirip dengan *branch*, tetapi tidak ada syarat
- Diperlukan tambahan sebuah *multiplexer* dengan sinyal kendali jump dan sebuah shift left 2
- Alamat jump sepanjang 32 bit merupakan konkat/gabungan dari:
  - PC+4 (31-28): 4 bit
  - *Immediate address jump*: 26 bit
  - Bit 00: 2 bit
  - Concat (shift_left_2(instruction(25-0)) , (PC+4 (31-28)))
- Contoh instruksi:
  - j target → j EXIT atau j 2500 # go to offset 10000

Instruksi Jump (2)

Fakultas Informatika
School of Computing
Telkom University



Instruksi Jump (3)

Contoh operasi instruksi Jump: j 10000

1. Geser kiri sebanyak 2 bit (dikalikan dengan 4) pada deretan bit alamat yang dituju sehingga menjadi 28 bit

   2500 (desimal) = 1001 1100 0100 menjadi 10 0111 0001 0000 = 10.000 (desimal)
2. Ambil 4 bit dari kiri hasil penjumlahan PC dengan 4
3. Gabungkan *(concate)* bit-bit pada nomor 1 dengan nomor 2 di atas
4. *Update* nilai PC dengan hasil penggabungan bit di atas

Daftar *Opcode*

0: tipe R (add, sub, and, or, slt)

2: jump (j)

4: branch (beq)

5: branch (bne)

8: penjumlahan dengan immediate (addi)

17: load upper dengan immediate (lui)

35: load (lw)

43: store (sw)

Instruksi dalam MIPS

Add – penambahan

```
add $s1,$s2,$s3  #jumlahkan isi reg s2 dengan isi reg s3
                 dan simpan hasilnya ke reg s1
```

Addi – add immediate

```
addi $sp,$sp, 4  #Jumlahkan isi reg sp dengan 4 dan
                 hasilnya simpan di dalam reg sp
```

ADDIU -- Add immediate unsigned

ADDU -- *Add unsigned*

AND -- Bitwise and

ANDI -- Bitwise and immediate

BEQ -- *Branch on equal*

BGEZ -- *Branch on greater than or equal to zero*

Instruksi dalam MIPS

BGEZAL -- *Branch on greater than or equal to zero and link*

BGTZ -- *Branch on greater than zero*

BLEZ -- *Branch on less than or equal to zero*

BLTZ -- *Branch on less than zero*

BLTZAL -- *Branch on less than zero and link*

BNE -- *Branch on not equal*

```
bne $t0,$zero, Less  #Lompat ke alamat Less jika isi
                     reg t0 tidak sama dengan nol
```

DIV -- *Divide*

DIVU -- *Divide unsigned*

Instruksi dalam MIPS

J -- *Jump*

JAL -- *Jump and link*

JR -- *Jump register*

LB -- *Load byte*

LI – Load immediate

li $v0, 5 #isi reg v0 dengan sign number (5)

LUI -- *Load upper immediate*

lui $t0, 255 #isi reg t0 bagian upper (bit 16-31) dengan 255

Isi memori **sebelum** instruksi dieksekusi:

001111	00000	01000	0000 0000 1111 1111

Isi memori **sesudah** instruksi dieksekusi:

0000 0000 1111 1111	0000 0000 0000 0000

LW -- *Load word*

lw $s1,100($s2) #isi reg s1 dengan data dari memori pada alamat hasil jumlahan dari isi reg s2 dengan 100 (offset)

MFHI -- *Move from HI*

MFLO -- *Move from LO*

Instruksi dalam MIPS

MULT -- *Multiply*

MULTU -- *Multiply unsigned*

NOOP -- *no operation*

OR -- *Bitwise or*

ORI -- *Bitwise or immediate*

SB -- *Store byte*

SLL -- *Shift left logical*

SLLV -- *Shift left logical variable*

SLT -- *Set on less than (signed)*

```
slt $t0,$s0,$s1  #reg t0 diisi dengan 1 jika isi reg s0
                         lebih kecil daripada reg s1
biasanya diikuti dengan perintah: bne $t0,$zero,Less
```

SLTI -- *Set on less than immediate (signed)*

```
slti $t0,$s0, 10  #reg t0 diisi dengan 1 jika isi reg s0
                         lebih kecil dari 10
biasanya diikuti dengan perintah: bne $t0,$zero,Less
```

SLTIU -- *Set on less than immediate unsigned*

SLTU -- *Set on less than unsigned*

Instruksi dalam MIPS

SRA -- *Shift right arithmetic*

SRL -- *Shift right logical*

SRLV -- *Shift right logical variable*

SUB – *Subtract*

```
sub $s1,$s2,$s3  #kurangkan isi reg s2 dengan isi reg s3
                 dan simpan hasilnya ke reg s1
```

SUBU -- *Subtract unsigned*

SW -- *Store word*

```
sw $s1,100($s2)  #simpan isi reg s1 ke memori pada
                 alamat hasil jumlahan dari isi reg s2
                 dengan 100 (offset)
```

SYSCALL -- *System call*

XOR -- Bitwise exclusive or

XORI -- *Bitwise exclusive or immediate*

Daftar Bit *Function*

opcode (6)	rs (5)	rt (5)	rd (5)	sa (5)	function (6)

Instruction		Function
add	rd, rs, rt	100000
addu	rd, rs, rt	100001
and	rd, rs, rt	100100
break		001101
div	rs, rt	011010
divu	rs, rt	011011
jalr	rd, rs	001001
jr	rs	001000
mfhi	rd	010000
mflo	rd	010010
mthi	rs	010001
mtlo	rs	010011
mult	rs, rt	011000
multu	rs, rt	011001
nor	rd, rs, rt	100111
or	rd, rs, rt	100101
sll	rd, rt, sa	000000
sllv	rd, rt, rs	000100

Instruction		Function
slt	rd, rs, rt	101010
sltu	rd, rs, rt	101011
sra	rd, rt, sa	000011
srav	rd, rt, rs	000111
srl	rd, rt, sa	000010
srlv	rd, rt, rs	000110
sub	rd, rs, rt	100010
subu	rd, rs, rt	100011
syscall		001100
xor	rd, rs, rt	100110

Daftar *Opcode*

For the `bgez`, `bgtz`, `blez`, and `bltz` instructions, the rt field is used as an extension of the opcode field.

opcode (6)	rs (5)	rt (5)	immediate (16)

Instruction		Opcode	Notes
addi	rt, rs, immediate	001000	
addiu	rt, rs, immediate	001001	
andi	rt, rs, immediate	001100	
beq	rs, rt, label	000100	
bgez	rs, label	000001	rt = 00001
bgtz	rs, label	000111	rt = 00000
blez	rs, label	000110	rt = 00000
bltz	rs, label	000001	rt = 00000
bne	rs, rt, label	000101	
lb	rt, immediate(rs)	100000	
lbu	rt, immediate(rs)	100100	
lh	rt, immediate(rs)	100001	
lhu	rt, immediate(rs)	100101	
lui	rt, immediate	001111	

Instruction		Opcode	Notes
lw	rt, immediate(rs)	100011	
lwc1	rt, immediate(rs)	110001	
ori	rt, rs, immediate	001101	
sb	rt, immediate(rs)	101000	
slti	rt, rs, immediate	001010	
sltiu	rt, rs, immediate	001011	
sh	rt, immediate(rs)	101001	
sw	rt, immediate(rs)	101011	
swc1	rt, immediate(rs)	111001	
xori	rt, rs, immediate	001110	

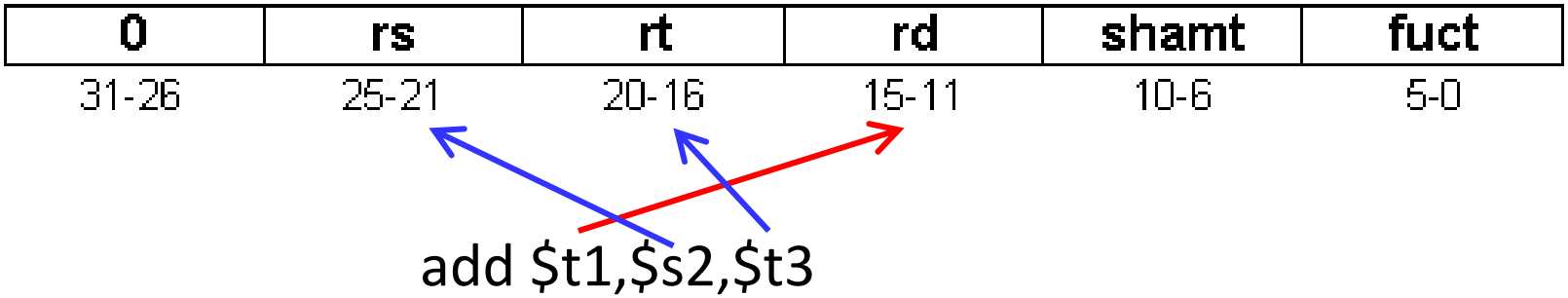
Mode Pengalamatan dalam MIPS

Mode Pengalamatan Pada MIPS

- Pengalamatan Register
- Pengalamatan *Base* atau *Displacement*
- Pengalamatan *Immediate*
- Pengalamatan *PC-relative*
- Pengalamatan *Pseudodirect*

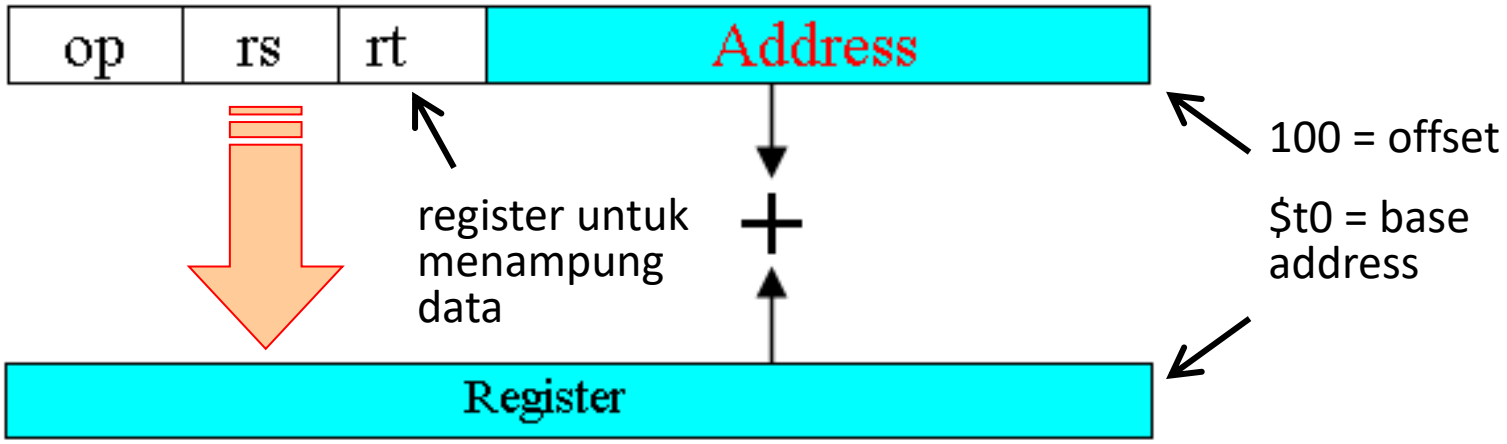
Pengalamatan Register

- Pengalamatan dengan mode register menggunakan register sebagai *operand*
- Register dapat berperan sebagai sumber atau tujuan
- Dalam MIPS terdapat 32 register
- Contoh: add $t1,$s2,$s3



Pengalamatan *Base* atau *Displacement*

- Pengalamatan dengan mode ini *operand*-nya berupa alamat *absolute memory* yang merupakan penjumlahan register base dengan *offset*
- Contoh: lw $s1, 100($t0)



Pengalamatan *Immediate*

- Dalam pengalamatan mode ini, *operand-nya* berupa bilangan konstanta pada instruksi tersebut
- Contoh instruksi:

  addi $s0,$s1,100 # $s0 = isi $s1 + 100

8	rs	rt	immediate
31-26	25-21	20-16	15-0

Contoh Pengalamatan *Immediate*

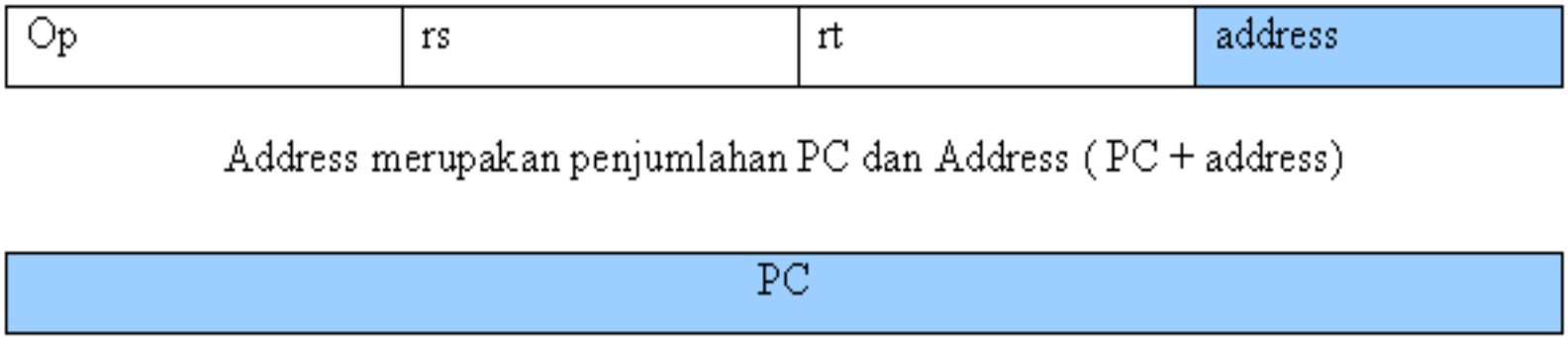

```
addi $t0,$t1,65      # $t0 = $t1 + 65
addi $sp,$sp,4
subi  $t0,$t0,7

addi $t0, $t0, -7
li $t3, 0x12345678    #$t3=0x12345678

lui  $at, 0x1234          # $at=0x1234 * 2^16
ori  $t3, $at, 0x5678  # $t3=$at|0x5678
bgez $t5, 16              # branch jika isi $t5>16
```

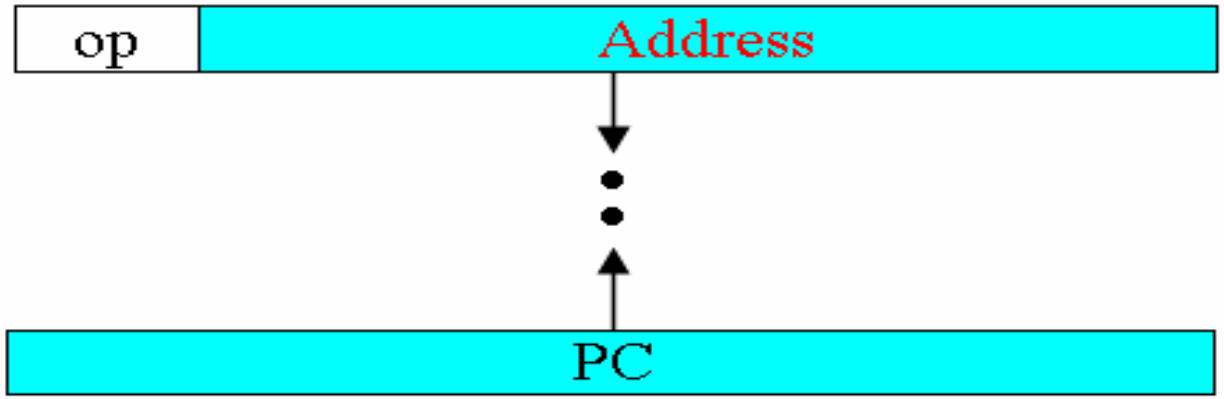
Pengalamatan *PC-relative*

- Dalam pengalamatan mode ini *operand* merupakan penjumlahan PC dengan bilangan konstan dalam instruksi
- Contoh: beq $s0, $s1, L1



Pengalamatan *Pseudodirect*

- Pada mode pengalamatan pseudodirect *operand*-nya berupa alamat jump (loncat) sebanyak 26 bit yang dikonkat dengan bit 31...28 pada PC
- Contoh: j L1



Referensi

- Hennessy, John L. dan Patterson, David A. 2018. "*Computer Organization and Design: The Hardware/Software Interface*". 5rd edition. Morgan Kaufmann publisher Inc. San Fransisco. USA
- http://chortle.ccsu.edu/AssemblyTutorial/ Chapter-01/