Organisasi dan Arsitektur Komputer (COA)
CII2A3

Arsitektur Prosesor MIPS

Disusun oleh: Tim dosen COA

Fakultas Informatika
Universitas Telkom

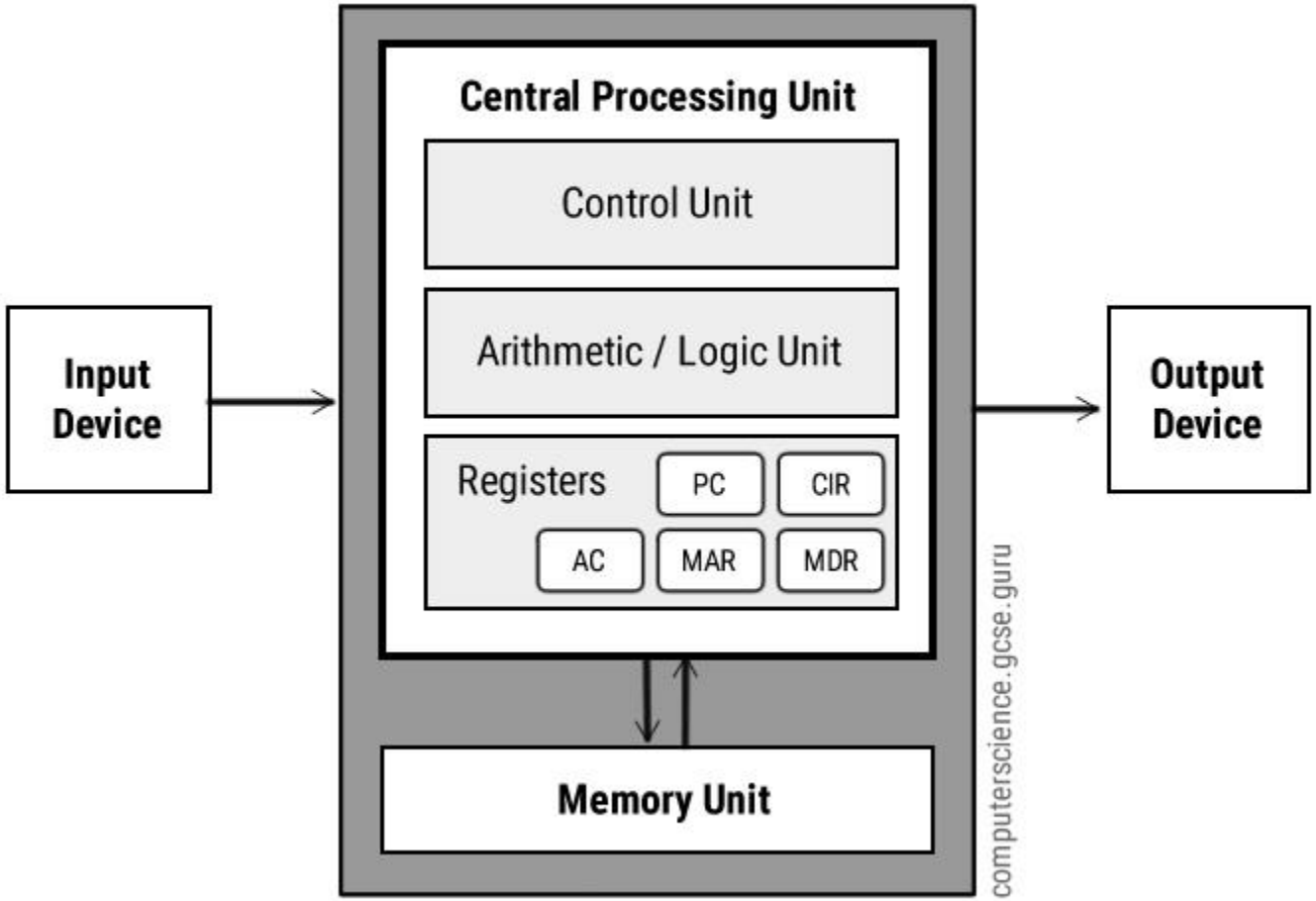
Desember 2022

Rencana Studi

Fakultas Informatika
School of Computing
Telkom University

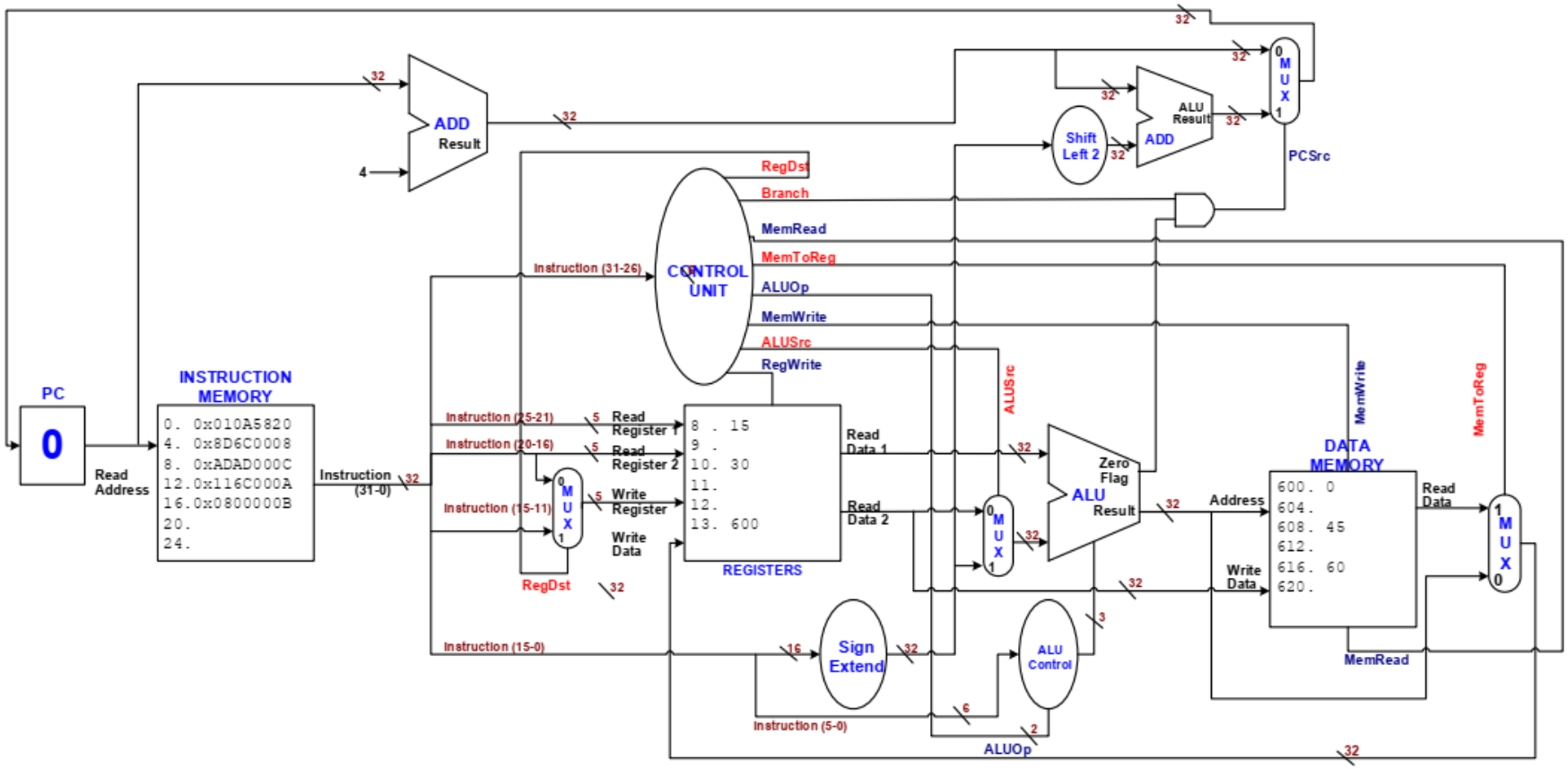
Rencana Studi		Rincian Nilai Kuis																Rincian Nilai Tugas										Rincian Nilai Hasil Proyek										Bobot Tiap CLO (%)			
Pertemuan Ke-	Materi	CLO 1			CLO 2					CLO 3				CLO 4				CLO 1			CLO 2						CLO	CLO 3						CLO 4									
		Kuis (Kognitif) (20 %)																Tugas Partisipatif (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				

Arsitektur Von Neumann



Arsitektur MIPS 1 Siklus

Fakultas Informatika
School of Computing
Telkom University



MIPS

- MIPS = *Million Instructions Per Second*
  = *Microprocessor without Interlocked Pipeline Stages*
- John L. Hennesy 1981
- Ide dasar: peningkatan kinerja prosesor dengan *pipeline*
- Pengeksekusian sebuah instruksi dibagi dalam beberapa *step*
- Instruksi dieksekusi secara independen
- Problem: *interlock*

Komponen Utama MIPS

Komponen Utama MIPS

Fakultas Informatika
School of Computing
Telkom University

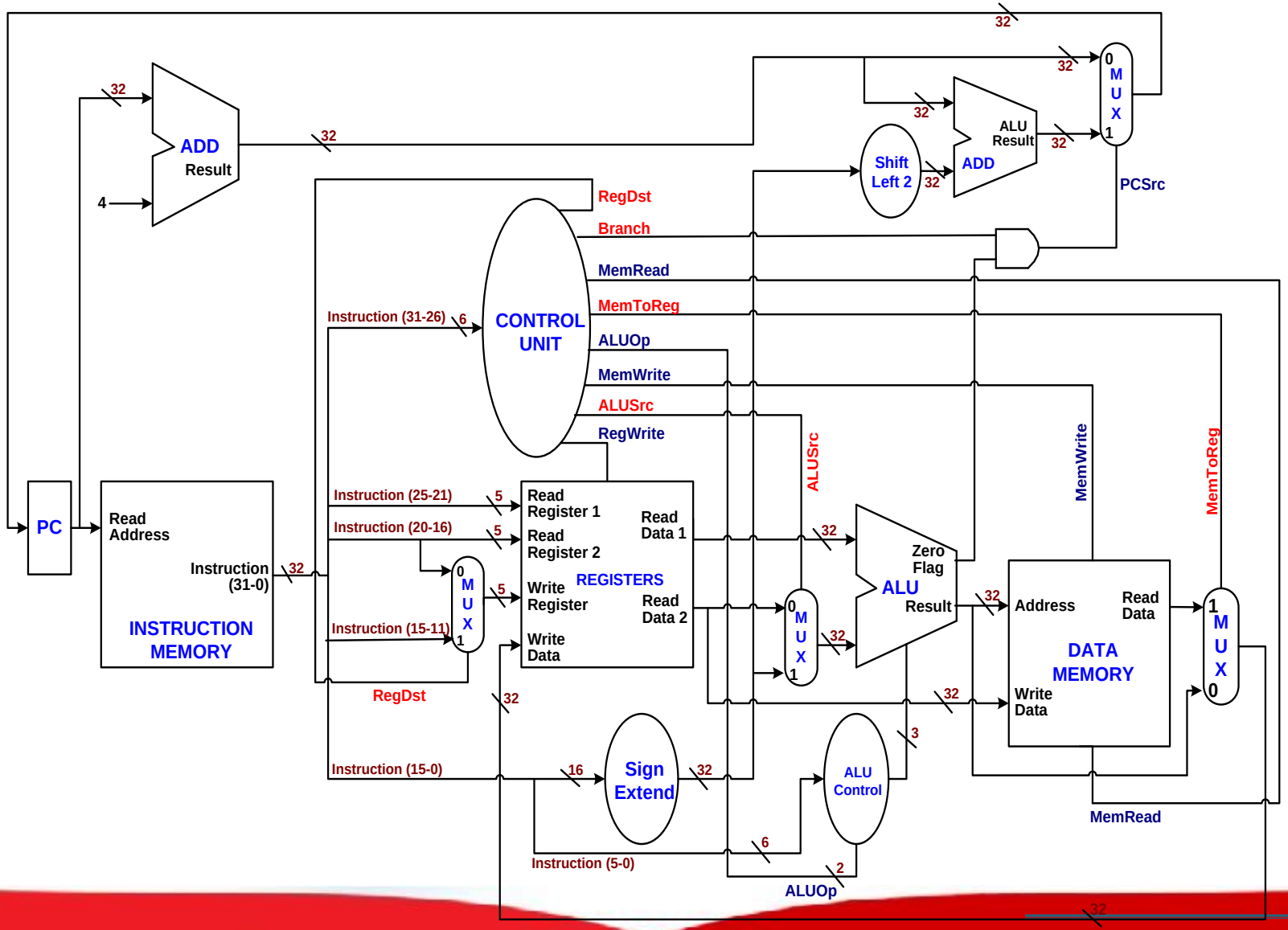
1. *Control Unit*: Bagian pengendali
2. *Program Counter (PC)*: Pencacah program
3. *Instruction Memory*: Memori instruksi
4. *Data Memory*: Memori data
5. *Register*: File register
6. *ALU (Arithmetic and Logical Unit):* Bagian pemroses aritmetik dan logika

Komponen Pendukung MIPS

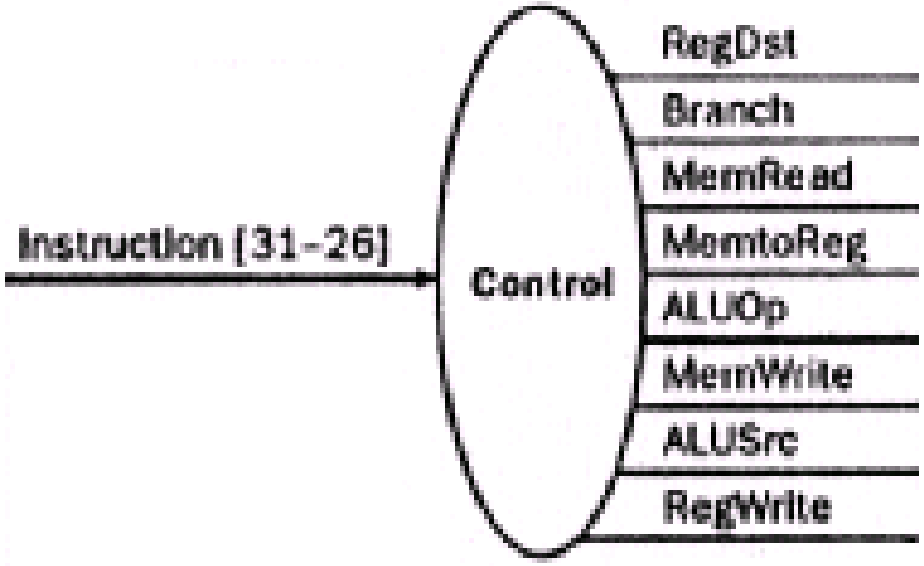
Fakultas Informatika
School of Computing
Telkom University

1. *Adder*: Penjumlah
2. *Shifter*: Penggeser
3. *MUX (Multiplexer):* Multiplekser
4. *Sign Extend*: menambah jumlah bit
5. *ALU control*: menentukan operasi ALU

Datapath MIPS



Bagian Pengendali *(Control Unit)* (1)



- Tujuan: Mengendalikan semua aktifitas prosesor
- Masukan: Kode operasi *(Opcode)* = 6 bit
- Keluaran: kendali semua komponen = 9 bit

Bagian Pengendali *(Control Unit)* (2)

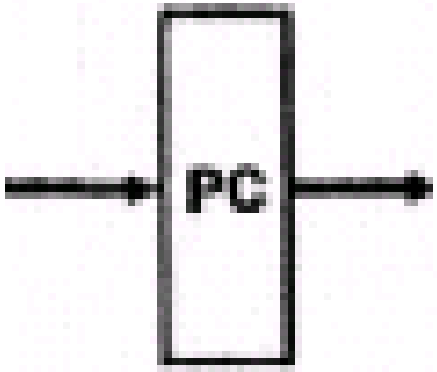
- Input pada bagian pengendali adalah kode operasi sebanyak 6 bit
- Kode operasi ini diterjemahkan (di-*decode*) untuk mengetahui jenis instruksinya
- Setelah mengetahui jenis instruksi tersebut, bagian pengendali mengeluarkan 9 bit kendali
- Bit kendali yang dikeluarkan oleh *Control Unit (CU)* akan menentukan operasi pada *datapath*

Bagian Pengendali *(Control Unit)* (3)

- Nama-nama kendali yang dihasilkan oleh CU:

1. RegDst (1 bit)
2. Branch (1 bit)
3. MemRead (1 bit)
4. MemToReg (1 bit)
5. ALUOp (2 bit)
6. MemWrite (1 bit)
7. ALUSrc (1 bit)
8. RegWrite (1 bit)

Pencacah Program (PC) (1)

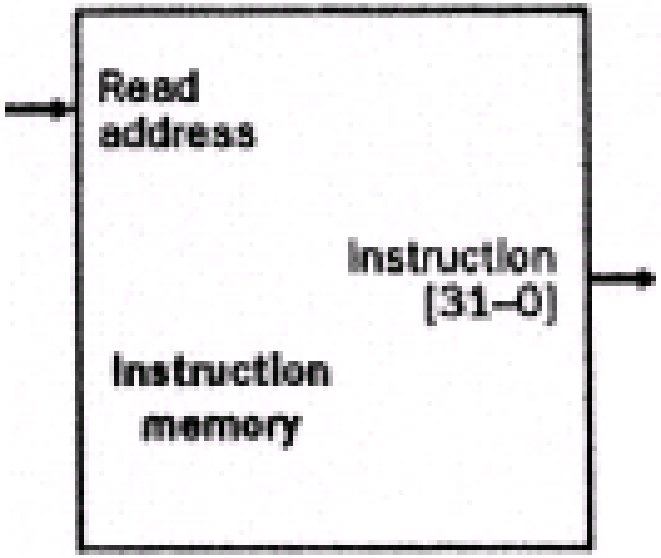


- Tujuan: Untuk menghitung alamat instruksi berikutnya yang akan dieksekusi
- Masukan: nilai PC = 32 bit
- Keluaran: alamat 32 bit pada memori instruksi

Pencacah Program (PC) (2)

- Pencacah program lebarnya 32 bit sehingga maksimal instruksi yang dapat diakses adalah 2^{32} buah
- Pencacah akan mengeluarkan nilai secara sekuensial dari 0x00000000 sampai 0x11111111
- Jika tidak ada instruksi pencabangan, maka nilai pencacah akan ditambah 4 setiap kali selesai instruksi

Memori Instruksi (1)

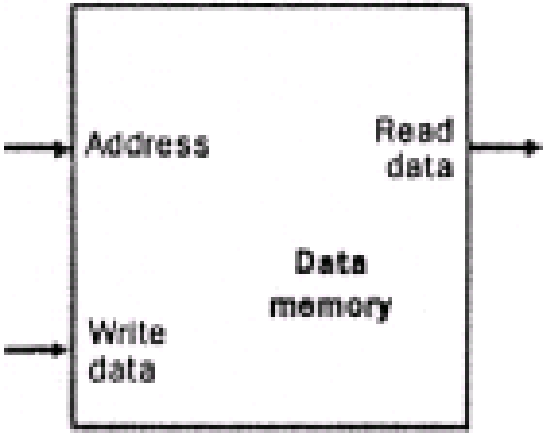


- Tujuan: Untuk menyimpan instruksi yang akan dieksekusi
- Masukan: alamat memori yang digunakan untuk menyimpan instruksi sebanyak 32 bit dari PC
- Keluaran: instruksi sebanyak 32 bit

Memori Instruksi (2)

- Menyimpan instruksi yang akan dieksekusi
- Lebar data pada setiap alamat 8 bit
- Lebar instruksi adalah 32 bit
- Setiap instruksi menempati 4 buah alamat dalam memori instruksi
- Instruksi yang telah dibaca masuk ke dalam bus dan diterjemahkan oleh bagian pengendali

Memori Data (1)

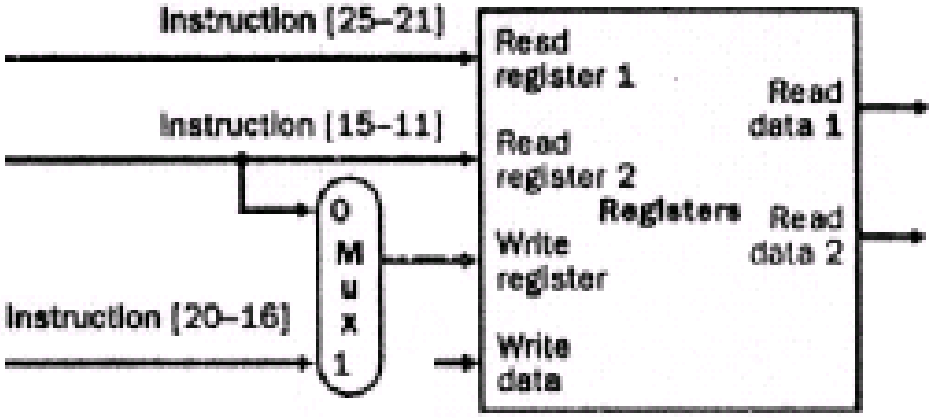


- Tujuan: menyimpan hasil penghitungan ALU
- Masukan:
  - Alamat memori yang akan digunakan untuk menyimpan data (akan ditulisi) sebanyak 32 bit
  - Data yang akan disimpan/ditulis sebanyak 32 bit
- Keluaran: Data yang dibaca sebanyak 32 bit

Memori Data (2)

- Lebar data setiap alamat adalah 8 bit
- Pada MIPS memori data dan memori instruksi sebenarnya disatukan
- Data biasanya disimpan pada alamat-alamat tinggi sedangkan instruksi pada alamat-alamat rendah/awal
- Memori data dikendalikan oleh jalur kontrol *MemRead* untuk membaca dan *MemWrite* untuk menulis

Register (1)



- Tujuan: menyimpan data yang akan dihitung oleh ALU dan menyimpan data hasil perhitungannya
- Masukan:
  - Alamat register 1 yang akan dibaca (5 bit)
  - Alamat register 2 yang akan dibaca (5 bit)
  - Alamat register yang akan ditulisi (5 bit)
  - Data yang akan ditulis ke resister (32 bit)
- Keluaran: data yang dibaca dari register 1 dan 2

Register (2)

- Register umum *(general)* jumlahnya 32 buah
- Masing-masing lebarnya 32 bit
- Menyimpan data hasil perhitungan ALU atau data yang berasal dari memori
- Data keluaran register menjadi masukan bagi ALU untuk dihitung
- Register-register diakses berdasarkan nomor registernya

Register (3)

- Penulisan register dikendalikan oleh jalur kendali *RegWrite*
- Pada pemrograman, nama register tidak diakses berdasarkan nomornya, tetapi berdasarkan namanya
- Masing-masing register diberi nama umum agar mudah diingat oleh pemrogram

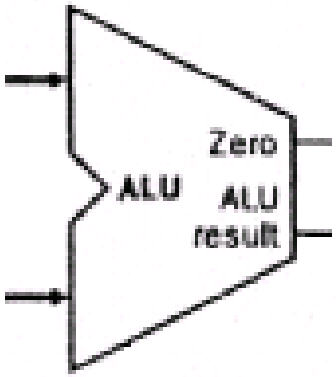
Register dan Fungsinya (1)

Nomor	Nama	Fungsi
$0	$zero	Register nol bernilai 0
$1	$at	assembler temporary
$2-$3	$v0-$v1	Fungsi untuk mengembalikan nilai
$4-$7	$a0-$a3	Sebagai fungsi argument
$8-$15	$t0-$t7	Temporary
$16-$23	$s0-$s7	Penyimpanan sementara
$24-$25	$t8-$t9	Temporaries
$26-$27	$k0-$k1	reserved for OS kernel
$28	$gp	global pointer
$29	$sp	Stack pointer
$30	$fp	Frame pointer
$31	$ra	return address (mengembalikan alamat)

Register dan Fungsinya (2)

- Register yang dipersiapkan untuk dipanggil dengan *syscall* (prosedur dan fungsi) tidak dapat diubah
- Sebagai contoh:
  - $s harus disimpan di dalam *stack* oleh prosedur jika ingin menggunakannya
  - $sp dan $fp selalu di-*increment* dengan konstanta kemudian *di-increment* kembali setelah prosedur selesai dilakukan
- Sementara itu $ra berubah secara otomatis dengan adanya *call function*

ALU
(Arithmatic and Logical Unit) (1)



- Tujuan: Mengolah (penjumlahan, pengurangan, logika) dua buah data masukan
- Masukan: input 1 (32 bit) dan input 2(32 bit)
- Keluaran: hasil pengolahan dan *zero flag*

ALU (2)

- MIPS adalah komputer 32 bit
- Jenis komputer ini ditentukan oleh lebar bus data yang masuk ke dalam ALU
- Selain mengeluarkan hasil penghitungan *(ALU Result)* ALU juga mengeluarkan *zero flag*
- *Zero flag* digunakan sebagai indikator apakah nilai keluarannya nol atau bukan
- Jika nilai keluarannya adalah nol maka *zero flag* bernilai 1 dan sebaliknya bernilai nol

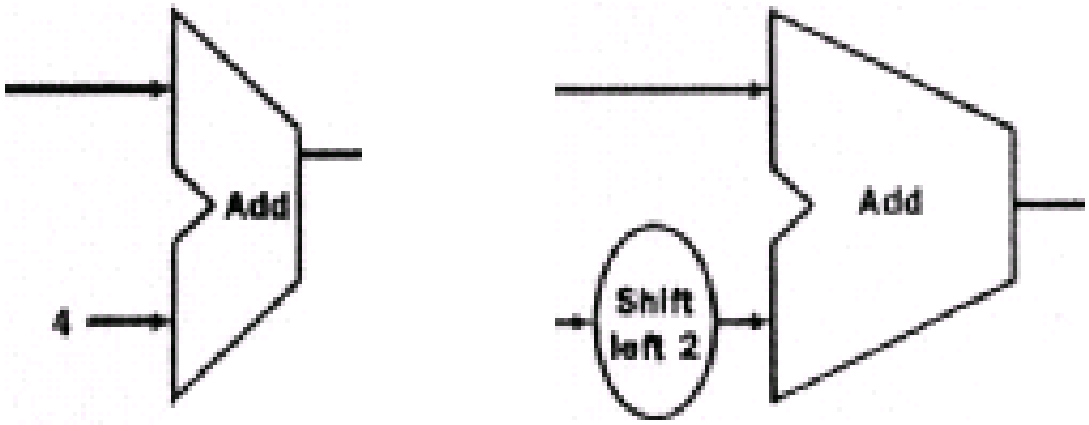
ALU (3)

- ALU memiliki 3 jalur kendali (3 bit):

Kode	Operasi
000	AND
001	OR
010	ADD
110	SUBSTRACT
111	SET ON LESS THEN

- Pada kelas instruksi R-format, ALU menjalankan salah satu dari kelima fungsi di atas, tergantung pada nilai 6-bit fungsinya
- Pada instruksi LW (*load word*) dan SW (*store word*) ALU digunakan untuk menghitung alamat memori dengan melakukan penjumlahan
- Pada instruksi beq (*branch on equal*) ALU melakukan proses pengurangan

Adder (1)

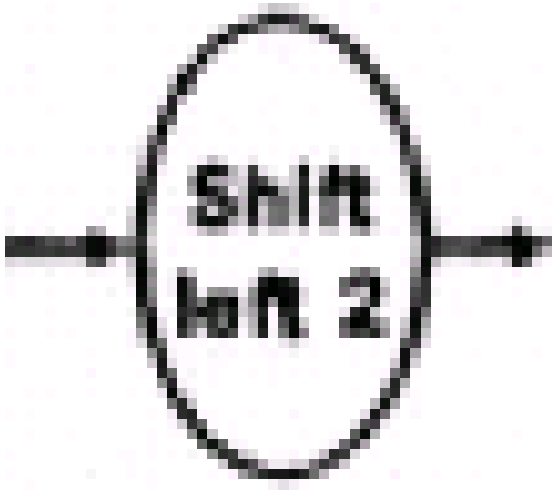


- Tujuan: Menjumlahkan dua buah *input*
- Masukan: dua buah *input* n bit
- Keluaran: sebuah n bit *output*

Adder (2)

- Rangkaian yang ada dalam *Adder* adalah *full adder*
- Simbol adder sama dengan simbol ALU, tetapi diberi nama Add
- Terdapat dua buah *adder:*
  - *Adder* yang menjumlahkan *input* dari PC (32 bit) dengan bilangan 4
  - *Adder* yang menjumlahkan hasil penjumlahan PC+4 (32 bit) dengan bilangan yang berasal dari bagian shift left 2 (32 bit)

Shifter (1)

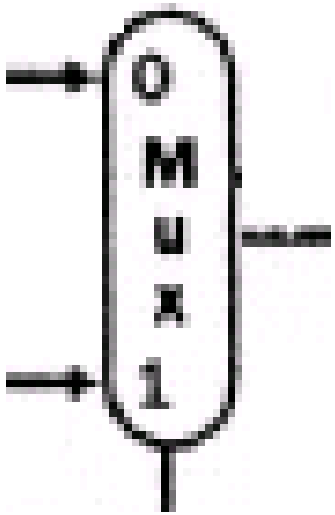


- Tujuan: menggeser bit-bit *input* ke kiri sebanyak 2 kali atau mengalikan *input* dengan 4
- Masukan: 1 *input* (32 bit)
- Keluaran: 1 *output* (32 bit)

Shifter (2)

- Shift left 2: Menggeser *input* ke kiri sebanyak 2 bit
- Operasi ini sama dengan mengalikan bilangan *input* dengan 4
- Contoh:
  - Masukan:
    0000000000000000 0000000000000010 → 2
  - Keluaran:
    0000000000000000 0000000000001000 → 8

MUX (Multiplexer) (1)

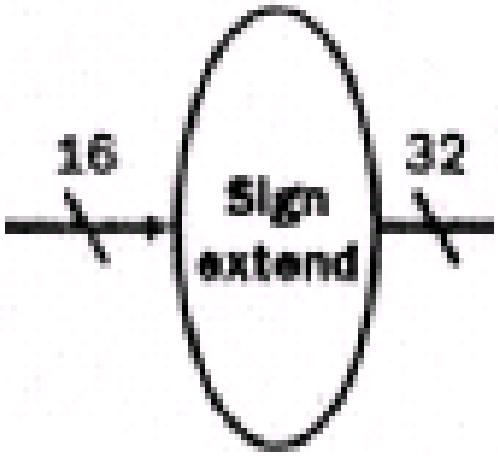


- Tujuan: Memilih satu dari 2 *input* yang tersedia untuk disalurkan ke *output*
- Masukan: 2 buah *input* masing-masing 32 bit dan sebuah *select* (1 bit)
- Keluaran: 1 *output* (32 bit)

MUX (Multiplexer) (2)

- Multiplexer ini disebut mux 2 ke 1
- Terdapat 2 buah *input* dan 1 buah *output*
- *Select* berfungsi menentukan input mana yang dipilih datanya untuk dikeluarkan
- Jumlah *select* tergantung banyaknya *input*
- Karena *input*-nya ada 2 maka jumlah *select*-nya cukup 1 buah saja yang dapat bernilai 0 atau 1

Sign Extend (1)



- *Tujuan*: Mengubah data 16 bit menjadi data 32 bit
- *Masukan*: 1 *input* sebanyak 16 bit
- *Keluaran*: 1 *output* sebanyak 32 bit

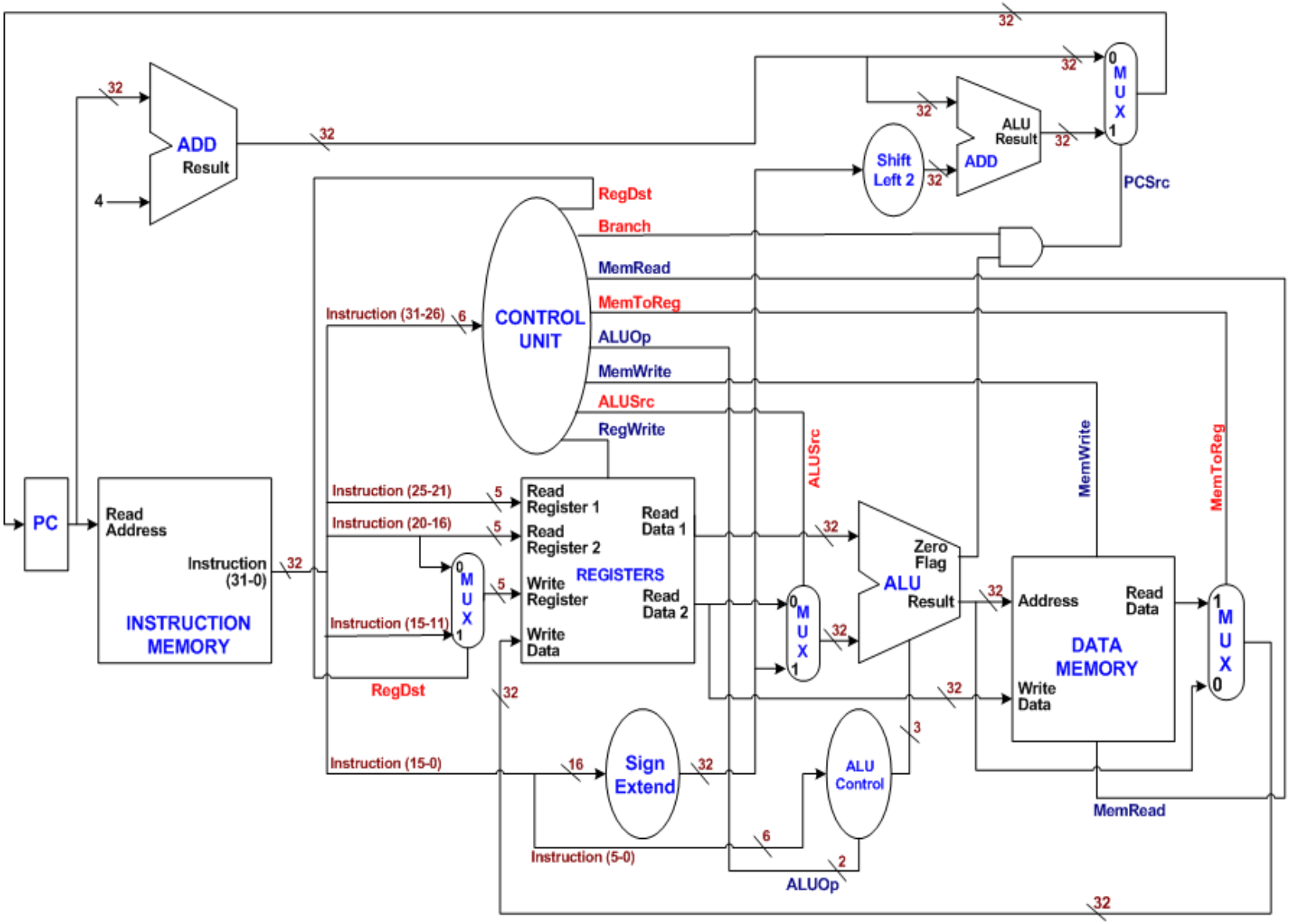
Sign Extend (2)

- Menambah bilangan 0 sebanyak 16 bit di awal (sebelah kiri) sehingga jumlah bit menjadi 32 bit
- Keluaran dari *Sign Extend* menjadi *input* ALU atau menjadi input *Shift left 2* yang terhubung ke *Adder*
- Contoh:
  - Masukan: 0000 0000 0010 1101
  - Keluaran:

    0000 0000 0000 0000 0000 0000 0010 1101

Unit Kendali

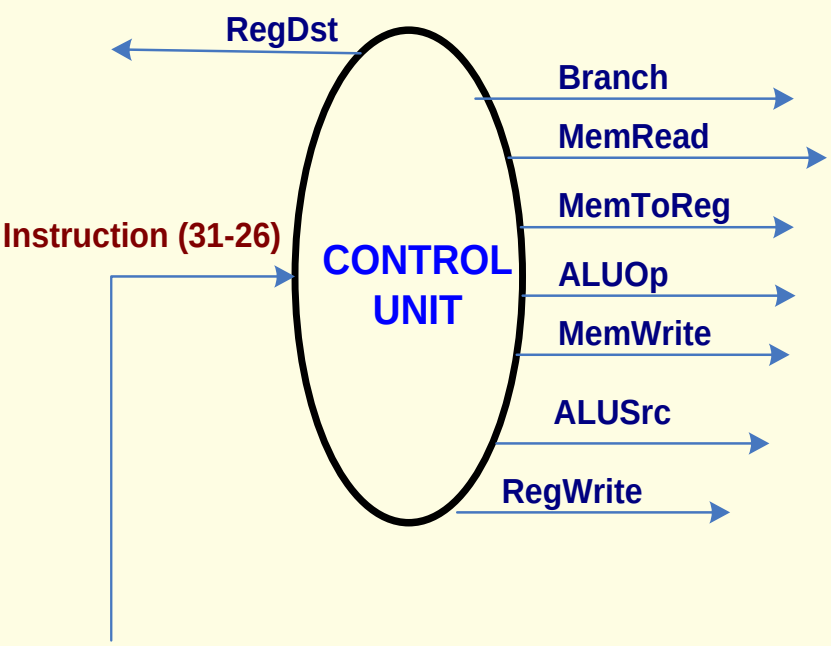
MIPS Datapath



Unit Kendali (1)

- Tujuan: mengendalikan semua aktifitas prosesor, atau lebih tepatnya untuk mengendalikan semua komponen seperti ALU, PC, Register, dll
- Masukan: *Operation Code (Opcode)* = 6 bit
- Keluaran (9 bit):
  1. RegDst
  2. Branch
  3. MemRead
  4. MemtoReg
  5. ALUOp (2 bit)
  6. MemWrite
  7. ALUSrc
  8. RegWrite

Unit Kendali (2)



- Masukan/*input* instruksi bit 31-26 adalah kode operasi
- Unit kendali akan menterjemahkan kode operasi dan mengeluarkan 9 bit kendali yang mengendalikan jalannya prosesor

Input Unit Kendali (1)

- Instruksi 32 bit terdiri dari *opcode* dan *operand*:
  - 6 bit di awal (dari MSB atau dari kiri) disebut opcode
  - 26 bit berikutnya adalah *operand*
- Contoh: ($ = register; s = source; t = target; d = destination)

Deskripsi:	Untuk menambahkan 2 nilai register dan menyimpannya di salah satu register juga
Operation:	$d = $s + $t
Syntax:	add $d, $s, $t
Encoding:	000000 ssss tttt ddddd 00000 100000 000000: merupakan opcode untuk operasi aritmatika. 100000: merupakan kode fungsi operasi penambahan. ssss : diisi dengan nilai dari $s tttt: diisi dengan nilai dari $t ddddd : diisi dengan nilai dari $d(tempat penyimpanan hasil operasi)

Input Unit Kendali (2)

Fakultas Informatika
School of Computing
Telkom University

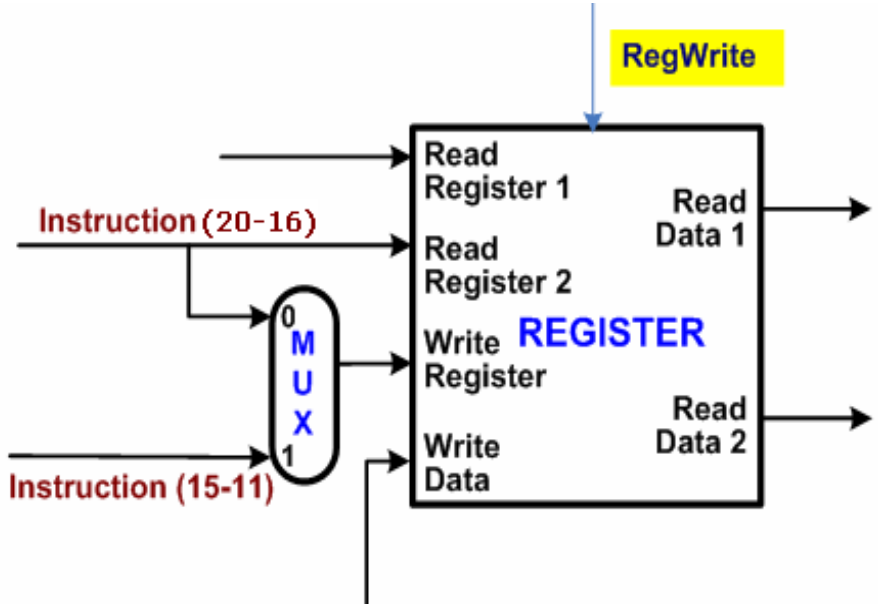
- *Operation Code* (*Opcode* 6 bits) = jenis instruksi yang akan dieksekusi
- Misal: instruksi add $t2, $s2, $t1
  - Artinya: jumlahkan data dalam register $s2 dan $t1 dan simpan hasilnya ke dalam register $t2
  - Instruksi dalam biner:

	6 bit	5 bit	5 bit	5 bit	5 bit	6 bit
bit ke:	31-26	25-21	20-16	15-11	10-6	5-0
	000000	10010	01001	01010	00000	100000

|Opcode|<----------Operand ---------->|

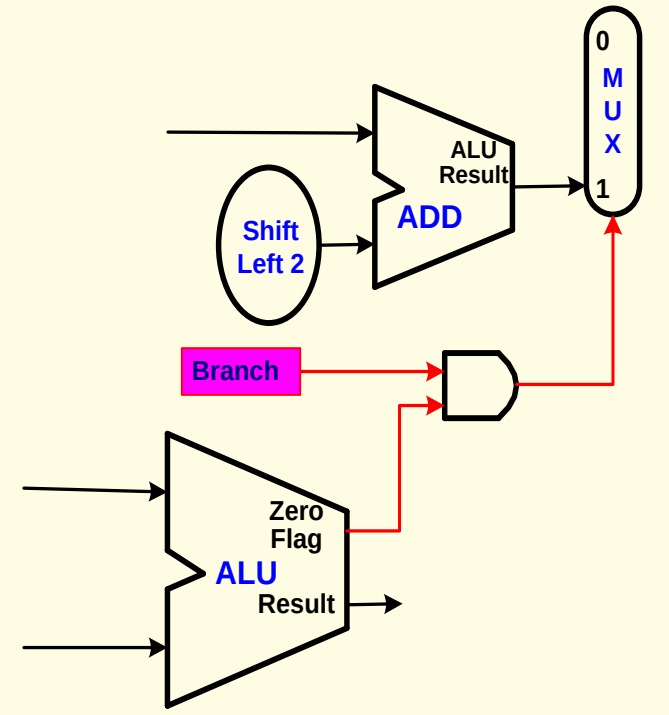
- 000000 adalah kode operasi untuk aritmatika (R)
- 10010 adalah nomor $s2 = 18
- 01001 adalah nomor $t1 = 9
- 01010 adalah nomor $t2 = 10
- 00000 tidak digunakan dalam operasi aritmatika
- 100000 adalah kode fungsi operasi add

Output CU: *RegDst*



- Tujuan: untuk mengendalikan multiplekser yang menentukan nomor register yang akan ditulisi (Write Register)
- Mux *Input*: bit instruksi 5 bit [20-16] atau [15-11]
- Mux *Output*: 5 bit tergantung nilai RegDst
  - If RegDst = 0 then output = Instruction[20-16]
  - If RegDst = 1 then output = Instruction[15-11]

Output CU: *Branch* (1)

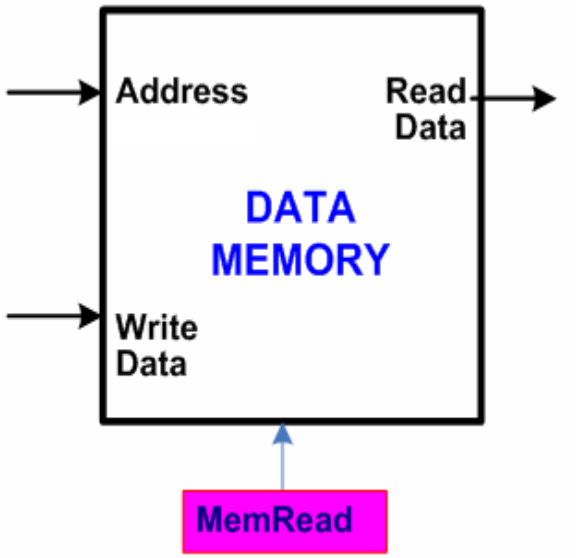


- Tujuan: Bersama dengan *zero flag* dari ALU digunakan untuk mengendalikan mux yang memilih sumber nilai PC selanjutnya
- If Zero flag=1 and Branch=1 then branch occur
- If not (zero flag=1 and branch=1) then branch not occur

Output CU: *Branch* (2)

- Jika *branch* tidak terjadi: (control branch = 0)
  - *Output* Mux berasal dari keluaran adder yang menjumlahkan PC dengan 4 (desimal)
  - PC = PC + 4
- Jika *branch* terjadi: (control branch = 1)
  - *Output* Mux berasal dari keluaran adder yang menjumlahkan (PC+4) dengan hasil dari Shift left 2
  - PC = (PC+4) + hasil dari Shift left 2

Output CU: *MemRead* (1)

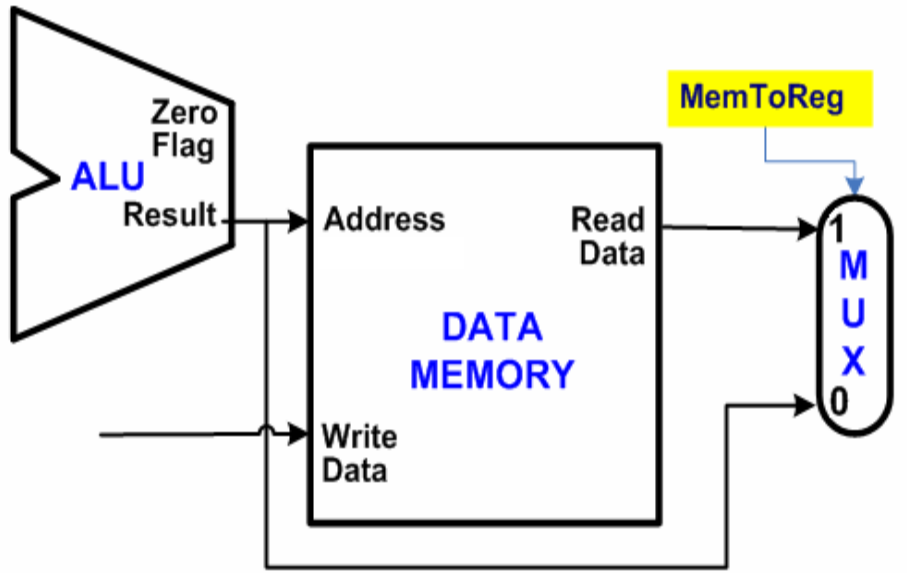


- Tujuan: Untuk mengaktifkan memori data agar data dari memori data dapat dibaca
- If *MemRead*=1 then Read Data ← Data Memory[address]

Output CU: *MemRead* (2)

- Memori bisa dibaca dan bisa ditulisi secara bergantian
- Pada saat memori dibaca, maka memori tidak dapat ditulisi, demikian pula sebaliknya
- *MemRead* mengendalikan pembacaan memori
- Jika *MemRead* bernilai 1, maka isi memori dapat dibaca melalui pin *Read Data*

Output CU: *MemToReg* (1)

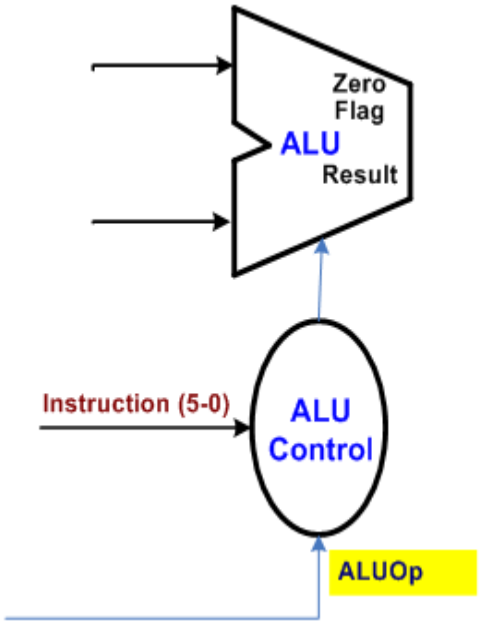


- Tujuan: Untuk mengendalikan mux yang menentukan asal data yang akan ditulis ke dalam register apakah dari memori data atau dari hasil ALU
- If MemtoReg=0 then Register←Data Memory
- If MemtoReg=1 then Register←ALUResult

Output CU: *MemToReg* (2)

- Jalur kendali yang mengendalikan *multiplexer* 2 ke 1
- MUX tersebut memilih salah satu *input* yang berasal dari ALU atau dari memori
- Jika *MemToReg* bernilai 1 maka data yang menuju Register berasal dari ALU
- Jika *MemToReg* bernilai 0 maka data yang menuju Register berasal dari Memori

Output CU: *ALUOp* (1)



- Tujuan: untuk memilih jenis operasi ALU yang akan dilakukan yang ditentukan oleh nilai 6 bit instruksi [5-0], sebagai fungsi dalam tipe R

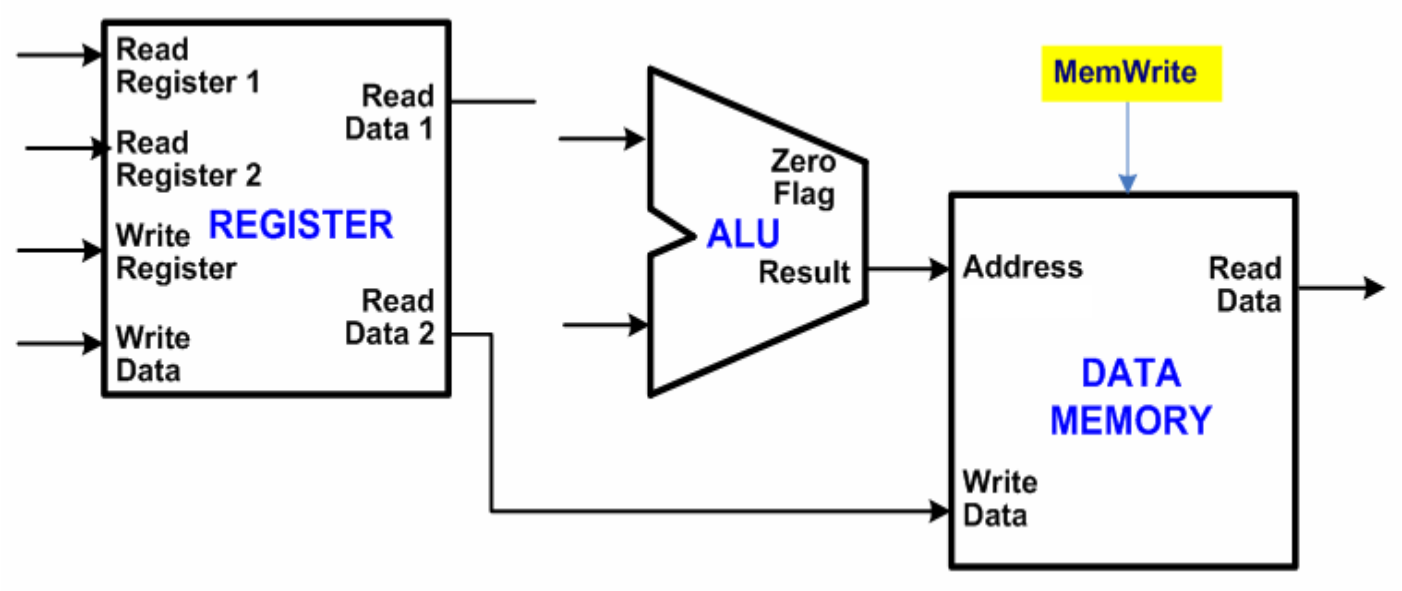
Output CU: *ALUOp* (2)

- AluOp adalah jalur kendali yang mengendalikan ALU control
- ALU control menentukan jenis instruksi yang dikerjakan oleh ALU
- Input dari ALU control berasal dari 6 bit fungsi *(Function field)* dari bit 0 – 5
- Pada instruksi tipe-R jenis operasinya ditentukan oleh keenam bit ini

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

- Apa maksud dari “set on less than”?

Output CU: *MemWrite* (1)



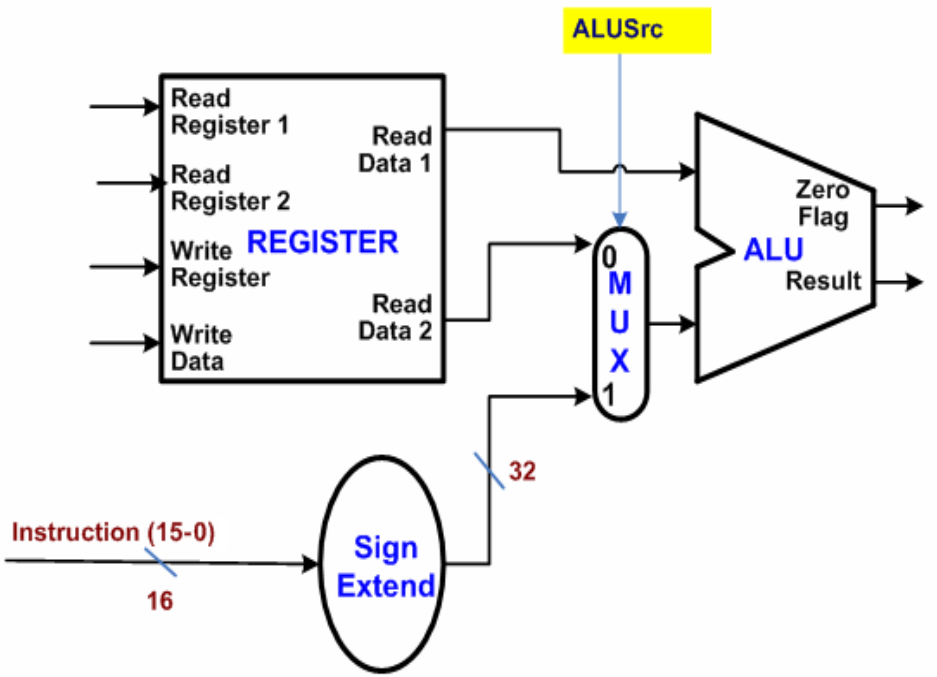
- Tujuan: Untuk mengaktifkan memori data agar dapat ditulisi dengan data yang berasal dari register (Read Data 2)
- If MemWrite=1 then DataMemory[ALUResult]←Read Data 2

Output CU: *MemWrite* (2)

Fakultas Informatika
School of Computing
Telkom University

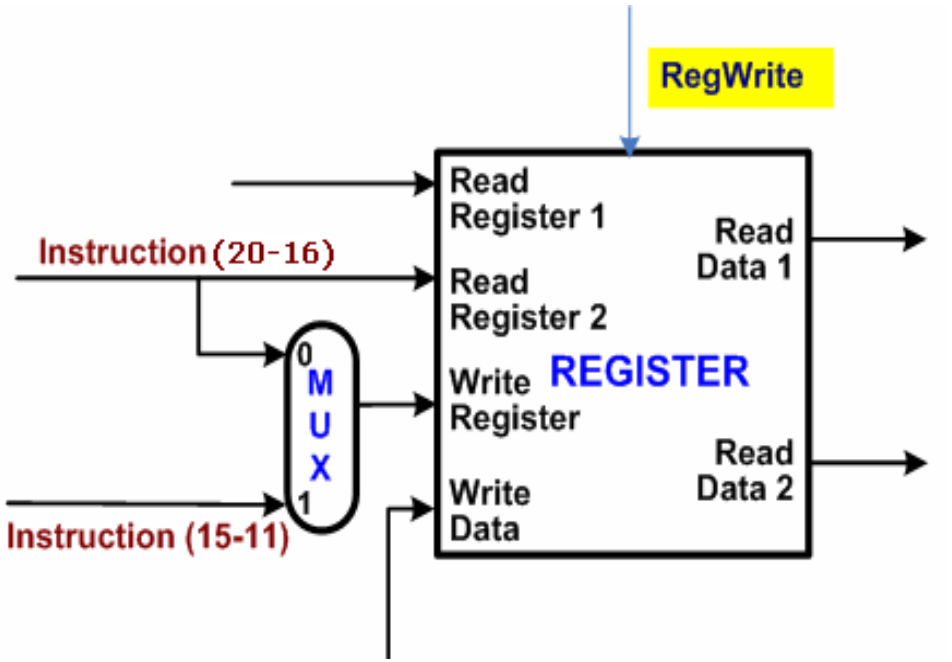
- *MemWrite* mengendalikan penulisan memori
- Alamat memori yang akan ditulisi ditentukan oleh hasil dari operasi ALU
- Data yang dimasukkan ke dalam memori berasal dari register (Read Data 2)

Output CU: *ALUSrc*



- Tujuan: Untuk mengendalikan mux yang menentukan asal data yang akan diolah oleh ALU apakah berasal dari register (Read Data 2) atau dari SignExtend(Instruction[15-0])
- If ALUSrc=0 then ALUInput←Register
- If ALUSrc=1 then ALUInput←SignExtend(Instruction[15-0])

Output CU: *RegWrite*



- Tujuan: Untuk mengaktifkan register agar register dapat ditulisi
- Data yang ditulisi bisa berasal dari memori atau dari hasil operasi ALU
- If RegWrite=1 then Register[writeRegister]←Write Data

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/