Tahmini Ders İçeriği
(Tentative Couse Schedule – Syllabus)

1. Hafta: Sayı sistemleri, onluk/ikilik taban sayı gösterimleri, mantıksal kapılar, computer system overview, başarım (performance)

2. Hafta: 2'lik tabanda işaretli sayılar, mikroişlemci tarihi, benchmarking, başarım,

3. Hafta: Başarım, Amdahl yasası, RISC-V development Environment, Verilog HDL ile Birleşik (Combinational) devreler

4. Hafta:, Verilog HDL ile sıralı (sequential) mantıksal devre ve sonlu durum makinası tasarımı, timing analysis

5. Hafta: Aritmetik devre tasarımları: Toplama, çıkarma, çarpma, bölme, trigonometri, square-root, hyperbolic, exponential, logarithm

6. Hafta: Fixed ve Floating-Point sayı gösterimleri

7. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

8. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

9. Hafta: Vize Çözümleri – CPU Tasarımına Giriş

10. Hafta: Tek-çevrim işlemci tasarımı (single-cycle CPU)

11. Hafta: Çok-çevrim işlemci tasarımı (multi-cycle CPU)

12. Hafta: Boruhatlı işlemci tasarımı (pipelined CPU)

13. Hafta: Bellek sistemi ve hiyerarşisi

14. Hafta: Gömülü sistemler, mikrodenetleyiciler, SoCs

RISC-V RV32I ISA CPU DESIGN

Amaç RV32I'da mevcut 47 instruction'un hepsini çalıştırabilen bir işlemci tasarımı

Tek dönem içerisinde gerçekleştirmek çok mümkün gözükmüyor, aşama aşama ilerlemek daha makul

İlk etapta implement edilecek instructionlar: **lw**, **sw**, **add**, **sub**, **and**, **or**, **beq** (Hennessy, Patterson)

Maksat, mikromimari (microarchitecture) tasarımı hakkında bilgi sahibi olmak

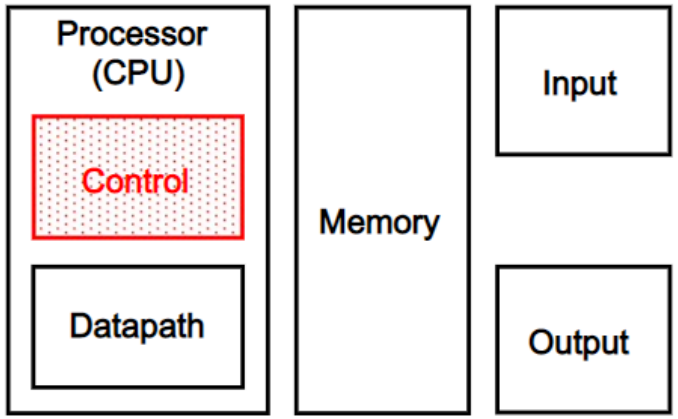
Mikromimari tasarımı, performansı ne ölçüde etkiliyor:

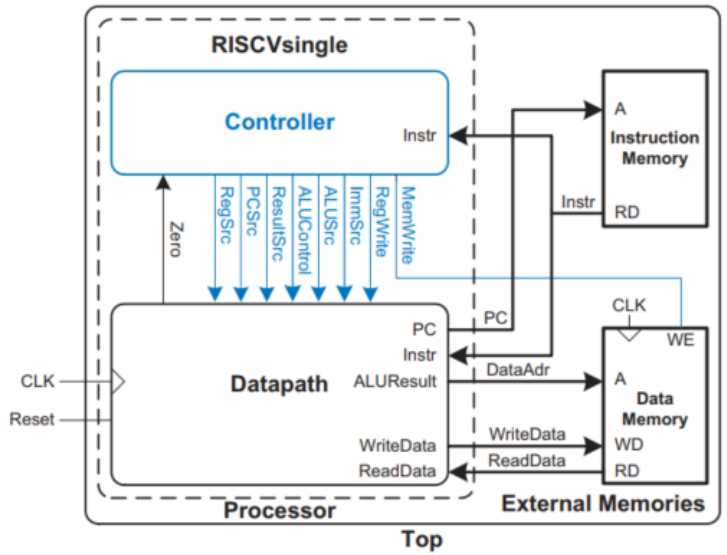
Compiler, bilgisayarın performans denkleminde buyruk sayısını ve CPI değerini etkiliyor.

Mikromimari ise CPI ve periyot (clock cycle time) parametreleri üzerinde etkili oluyor.

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

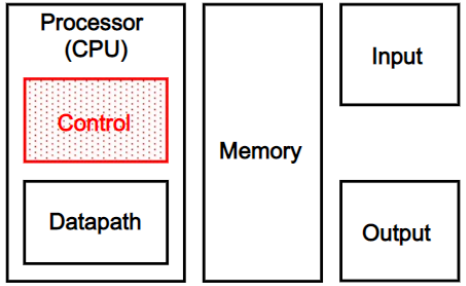




CPU DATAPATH DESIGN

İşlemci mikromimari tasarımı datapath (veriyolu) ve control (denetim) olarak ikiye ayrılır

Veriyolu kısmı blok tasarımı aşağıdaki gibidir:



PC: Program Counter, **ALU** Arithmetic & Logic Unit

lw ve **sw** instructionları **Data Memory**'ye erişim sağlayacaklar

Bütün instructionlar **ALU**'ya uğrayacak

add, **sub**, **and**, **or**, **lw** instructionları Register File'a yazma işlemi yapacak

PC register **beq** instruction sonrası +4 dışında bir değer alabilecek

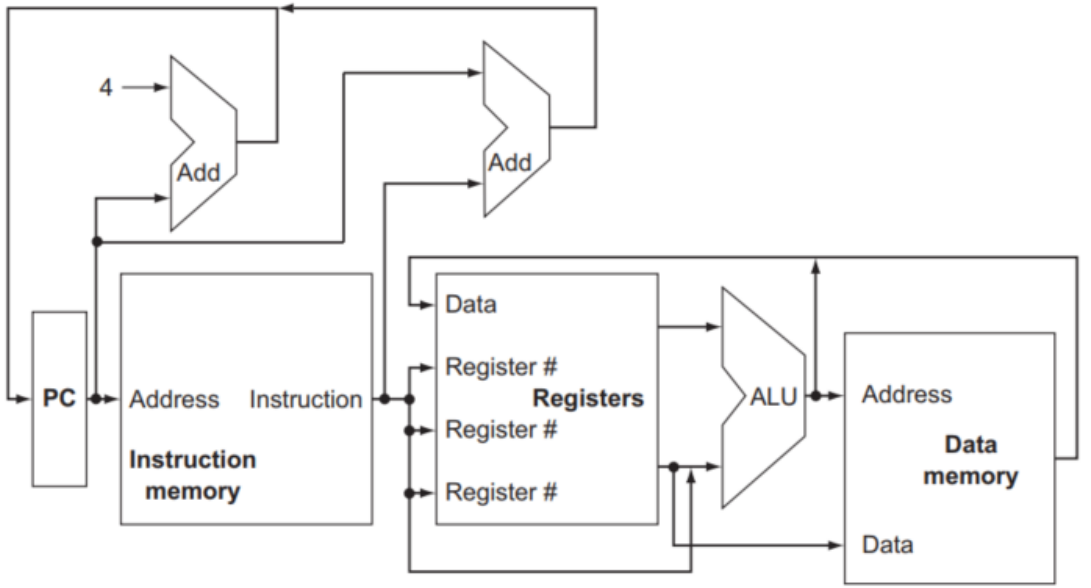
ALU'da **lw** ve **sw** instructionları için adres hesaplanacak

ALU'da **add**, **sub**, **and**, **or** için aritmetik ve logic işlemler yapılacak

ALU'de **beq** instruction için eşitlik kontrolü yapılacak

PC register değeri bir sonraki instruction adresini gösterecek

Register File, 2 okuma 1 yazma portu olan küçük bir bellek olacak

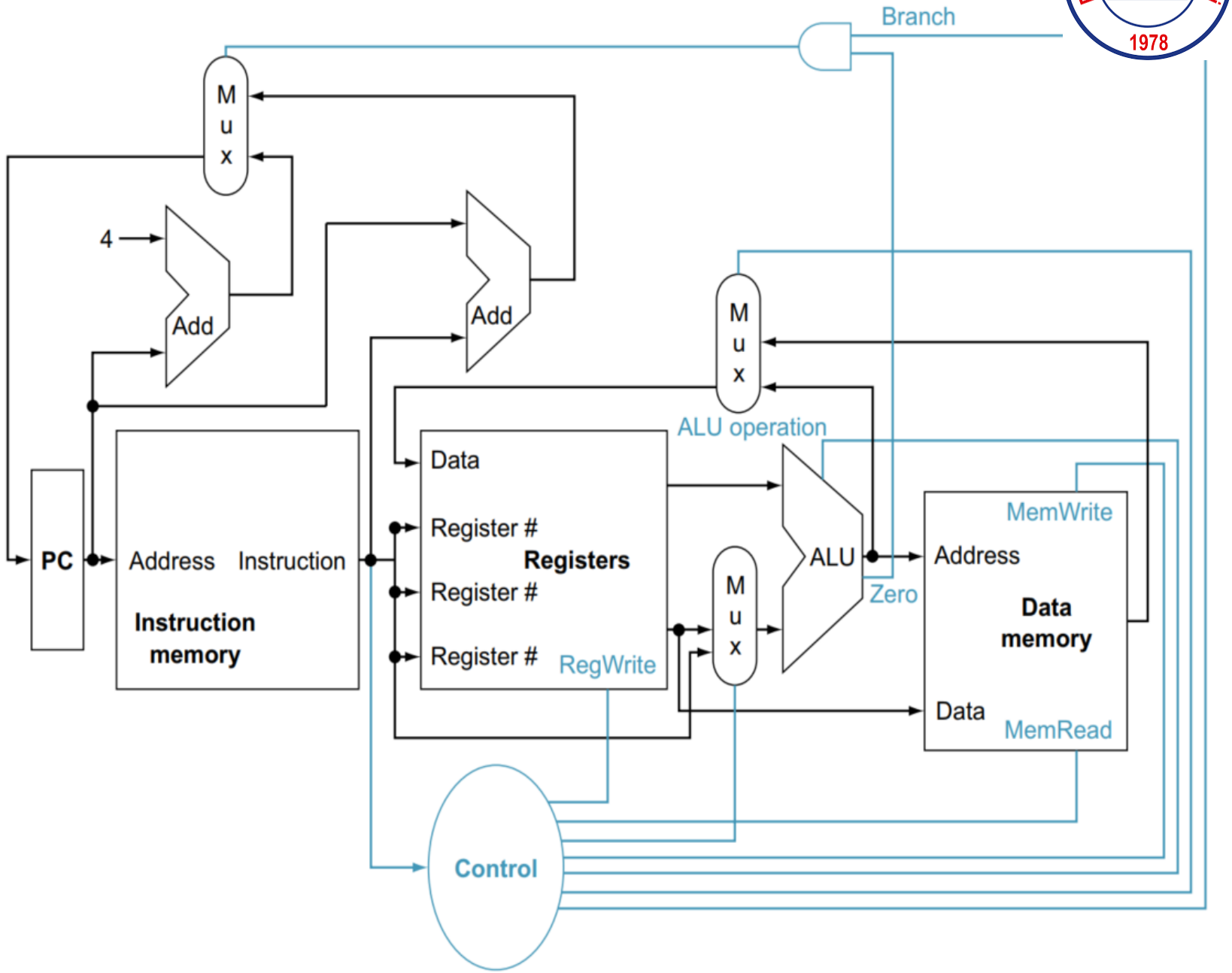


CPU CONTROL & DATAPATH DESIGN

Kontrol sinyalleri, instruction bellekten getirildikten sonra (fetch), opcode ve function alanlarının (field) çözümlenmesi ile (decode) oluşturulur.

ALU'daki **Zero** kontrol sinyali, **beq** instruction için, 2 register içeriğinin eşit olması durumunu bildirecektir.

Bu kontrol ve veriyolu tasarımı, 7 adet instruction için yeterli olmakla birlikte, desteklenecek instruction sayısı ve çeşidi arttıkça eklemeler yapılması gereklidir.



CPU DATAPATH DESIGN

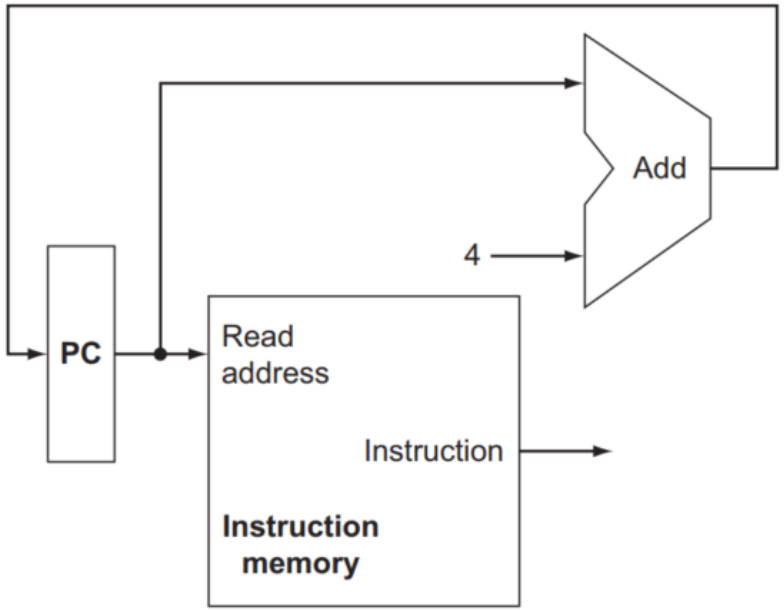
CPU'yu ilk aşamada tek saat vuruşunda çalışacak şekilde (single-clock cycle) tasarlayacağız

7 instruction'a baktığımızda beq dışında bütün instructionlar işletildiğinde PC, 4 arttırılmalıdır

Bunun nedeni, RISCV ISA'da bütün instructionları eşit uzunlukta ve 4 bayt olmasıdır

NOT: RISC yerince CISC ISA kullanılsaydı, PC her zaman 4 artmayacaktı, çünkü CISC'de instructionlar farklı uzunlukta olabiliyor

Veriyolunun ilk bileşenleri 32-bit register PC, yeterli büyüklükte read-only bir Instruction Memory ve Adder



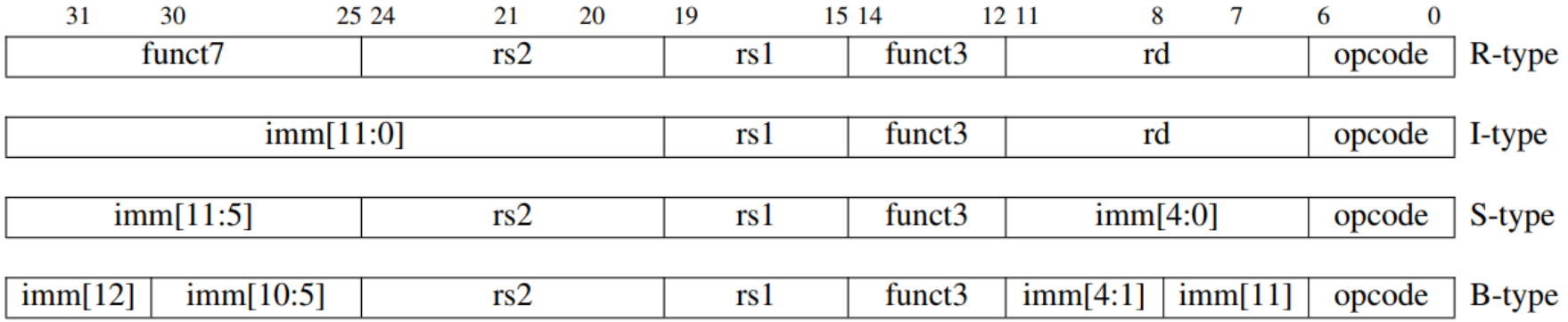
CPU DATAPATH DESIGN

add, sub, and, or → R-type

lw → I-type

beq → B-type

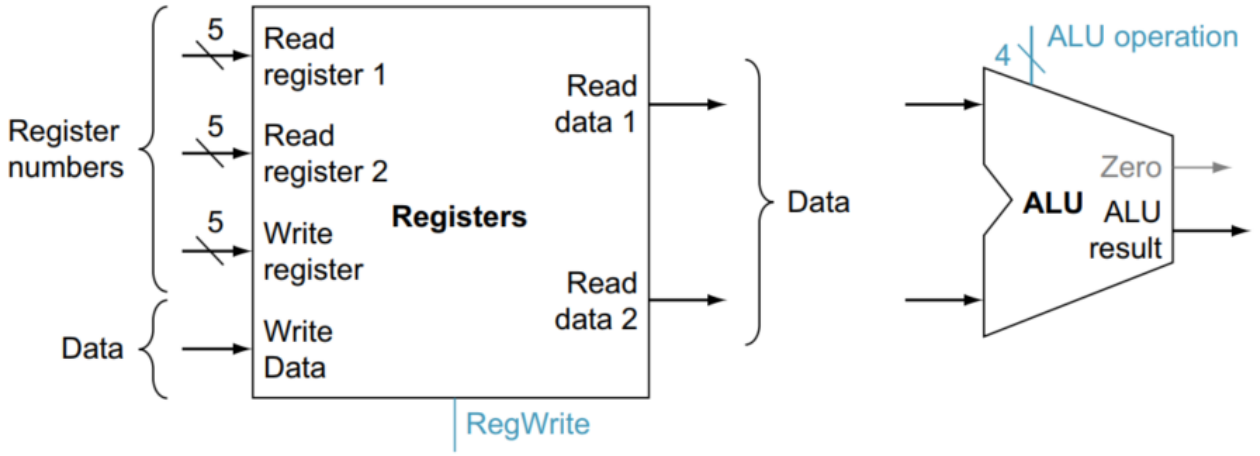
sw → S-type



R-type instructionlar işletilirken Register File'dan 2 adet register okunacak (rs1,rs2), ALU'da işlem gerçekleştirilip sonuç Register File'da hedef registerına yazılacak

Register file 32 adet 32-bit registerdan meydana gelecektir, 32 adet register olduğu için register adresleri 5-bit

ile ifade edilecek



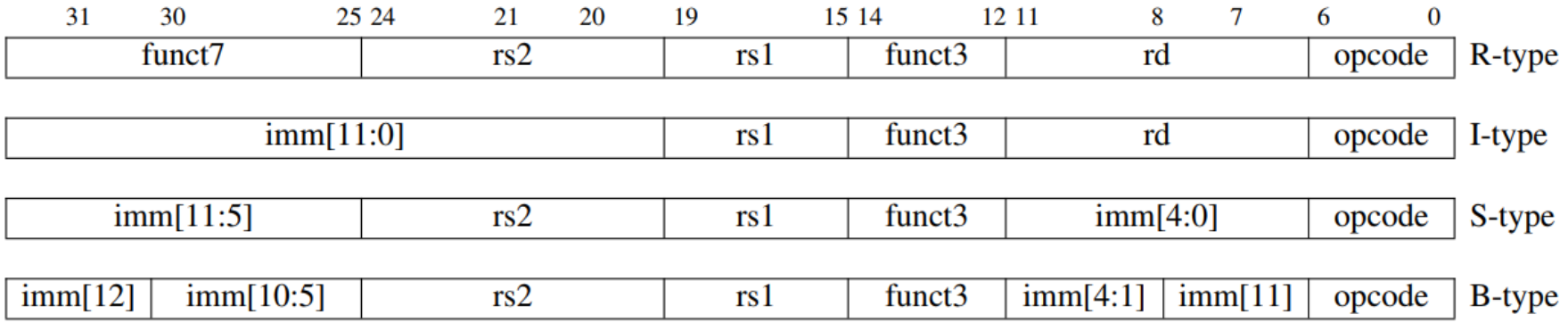
CPU DATAPATH DESIGN

add, sub, and, or → R-type

lw → I-type

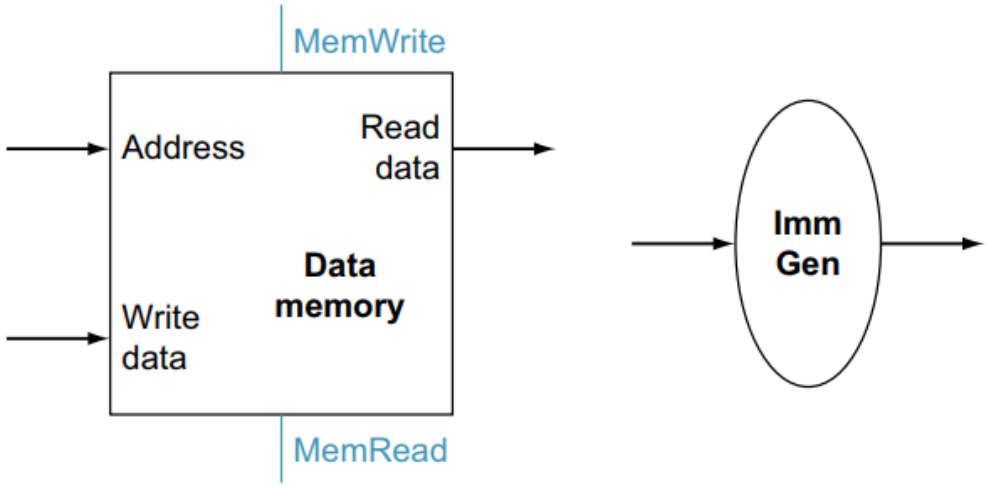
beq → B-type

sw → S-type



I-type instruction (lw) işletilirken Register File'dan 1 adet register okunacak (rs1) ve bu register içeriği ile immediate değeri toplanarak Bellek adresi hesaplanacak

Bellek adresindeki değer de Register File'da hedef registerına yazılacak (rd)



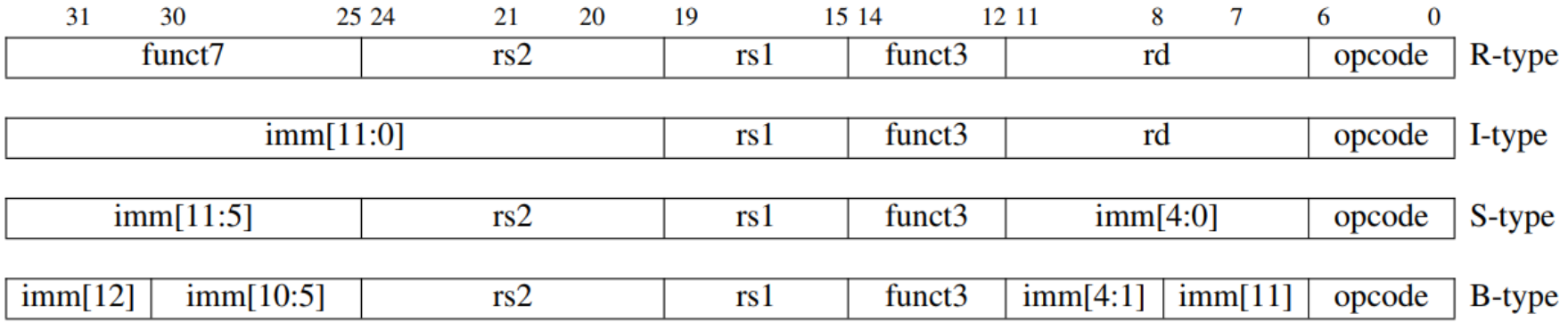
CPU DATAPATH DESIGN

add, sub, and, or → R-type

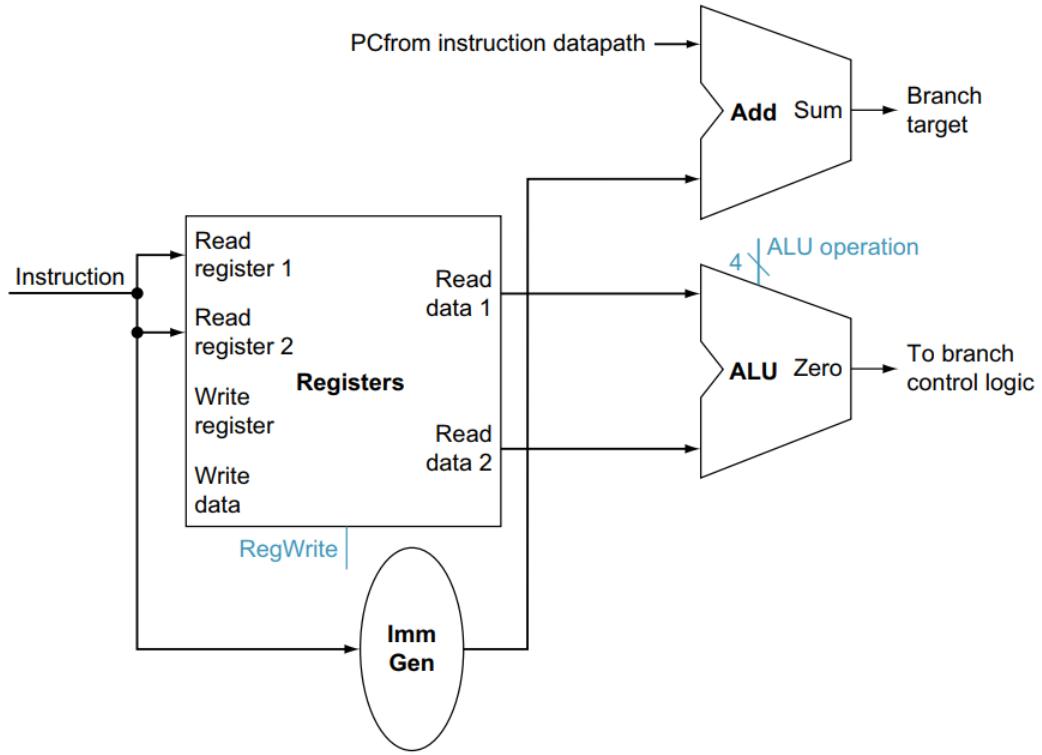
lw → I-type

beq → B-type

sw → S-type



B-type instruction (beq) işletilirken Register File'dan 2 adet register okunacak (rs1,rs2) karşılaştırma sonucuna göre (Zero) PC değeri ya +4 ya da +Branch Target olacak



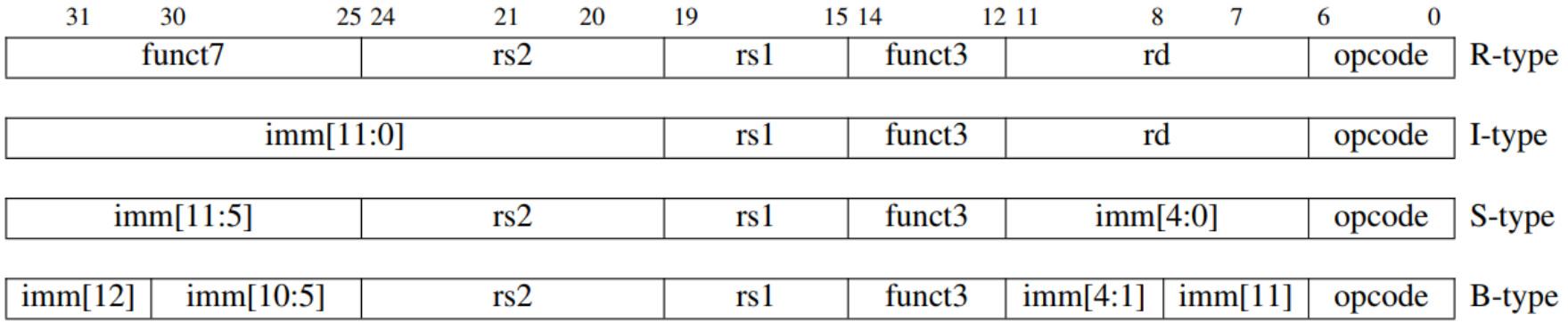
CPU DATAPATH DESIGN

add, sub, and, or → R-type

lw → I-type

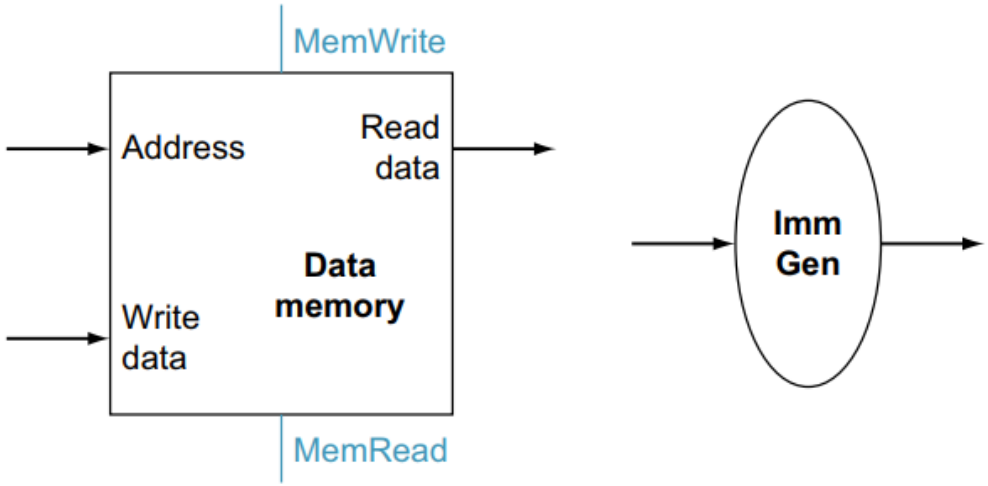
beq → B-type

sw → S-type

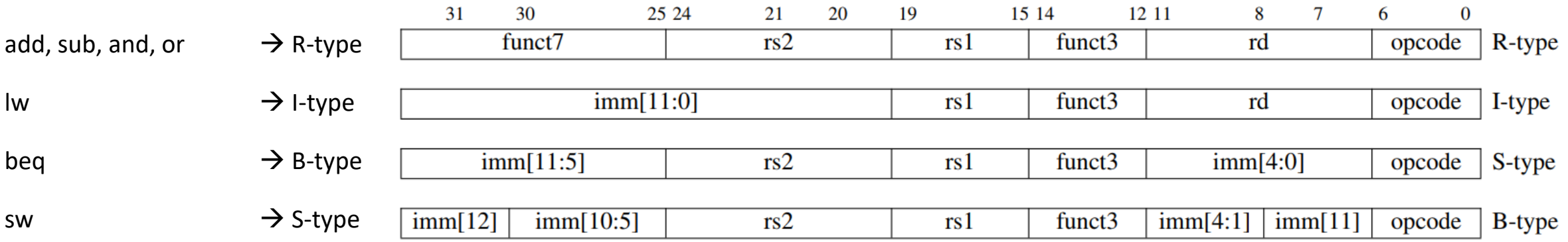


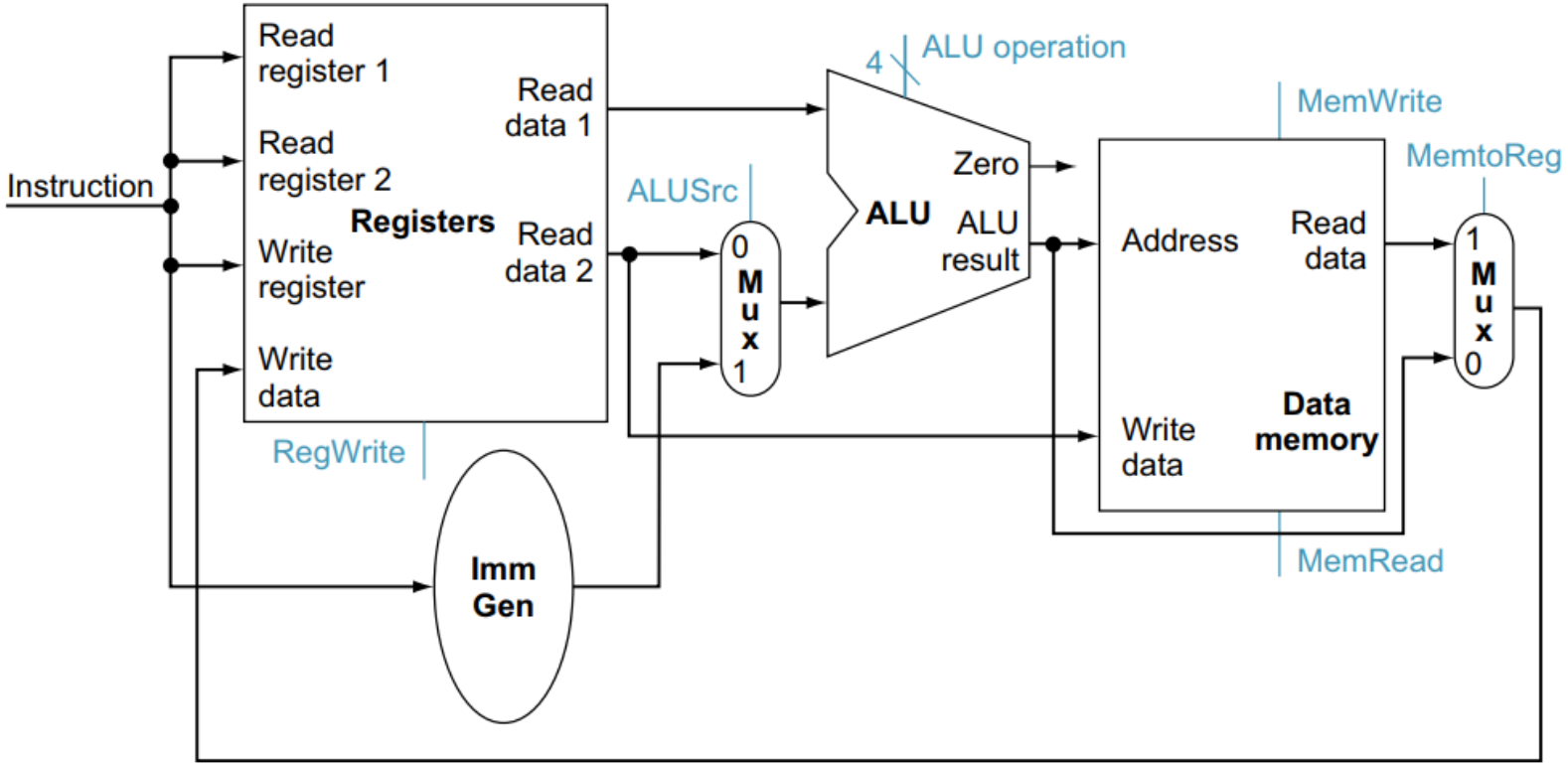
S-type instruction (sw) işletilirken Register File'dan 2 adet register okunacak (rs1,rs2) ve rs1 içeriği ile immediate değeri toplanarak Bellek adresi hesaplanacak

Bellek adresindeki değere diğer register (rs2) içeriği kaydedilecek

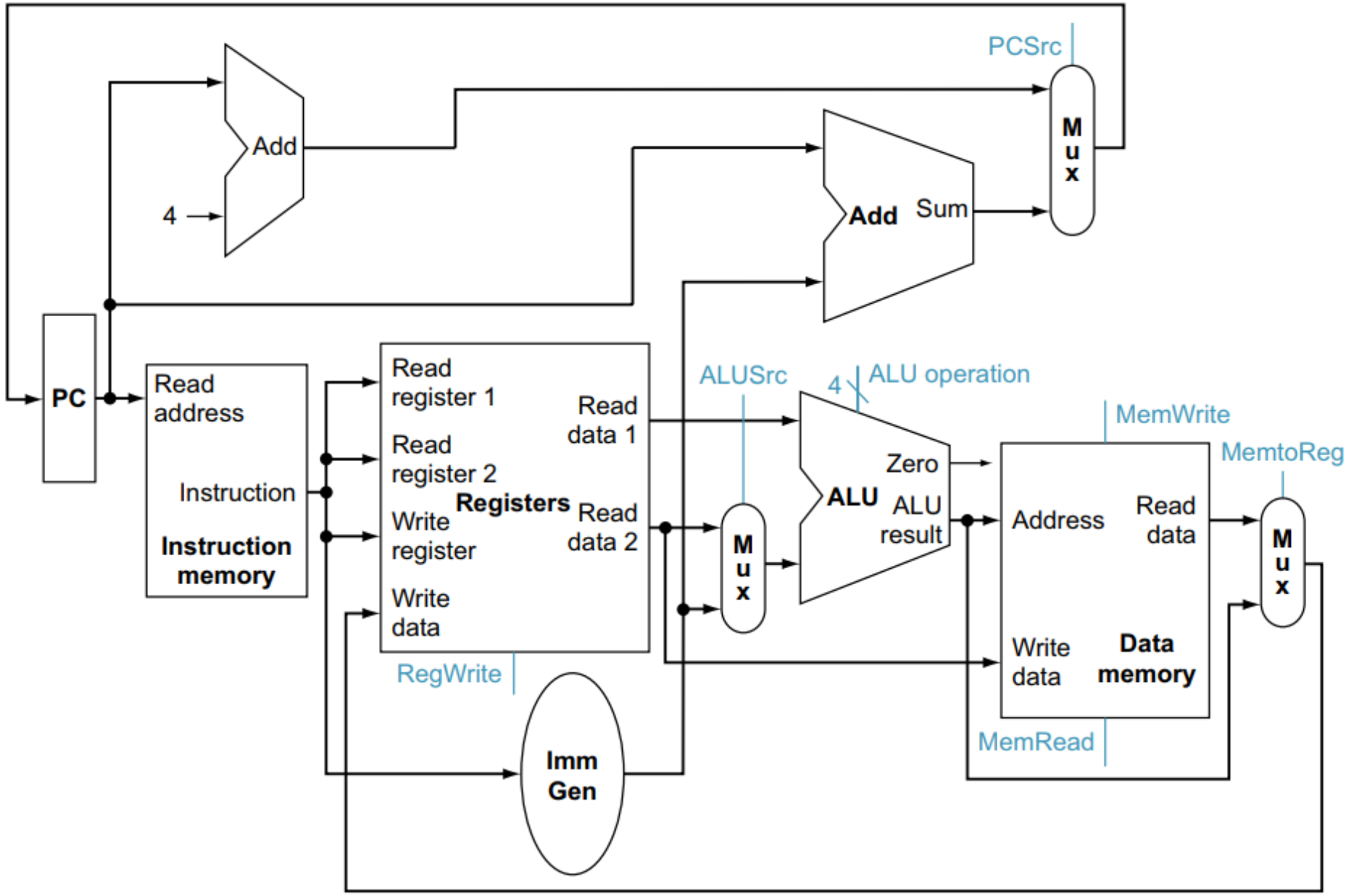


CPU DATAPATH DESIGN – Memory & R-Type





CPU DATAPATH DESIGN – All 7 Instructions



CPU CONTROL DESIGN

ALU operation

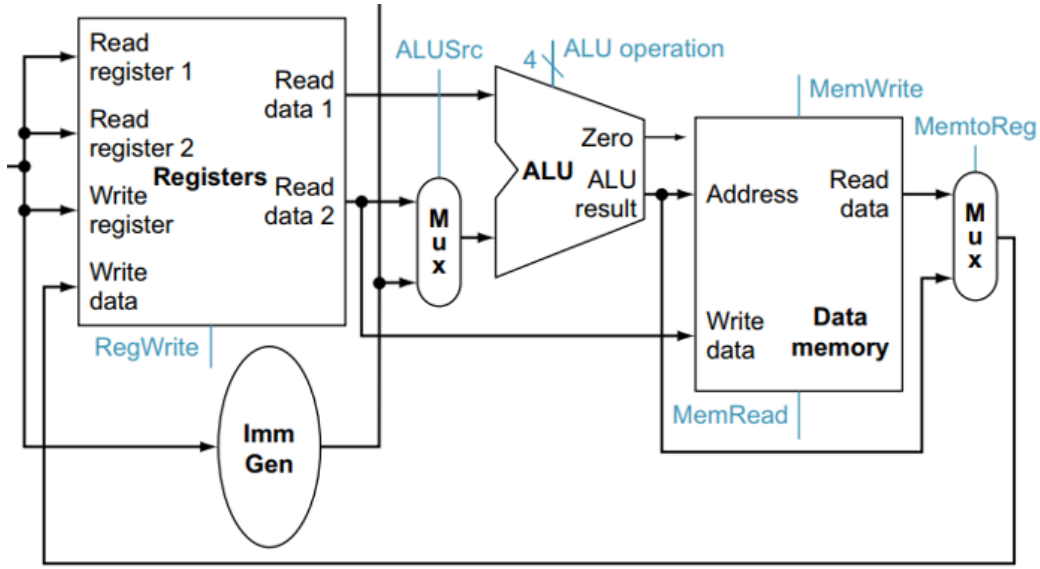
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract

beq, sub → subtract (0110)

lw, sw, add → add (0010)

and → AND (0000)

or → OR (0001)



imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[11:0]		rs1	010	rd	0000011	I lw
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	S sw
0000000	rs2	rs1	000	rd	0110011	R add
0100000	rs2	rs1	000	rd	0110011	R sub
0000000	rs2	rs1	110	rd	0110011	R or
0000000	rs2	rs1	111	rd	0110011	R and

beq, **lw** ve **sw** opcode'dan anlaşılıyor

add, **sub**, **or** ve **and** için func3 ve func7'ye bakılması lazım

or ve **and** func3'ten ayrışıyor

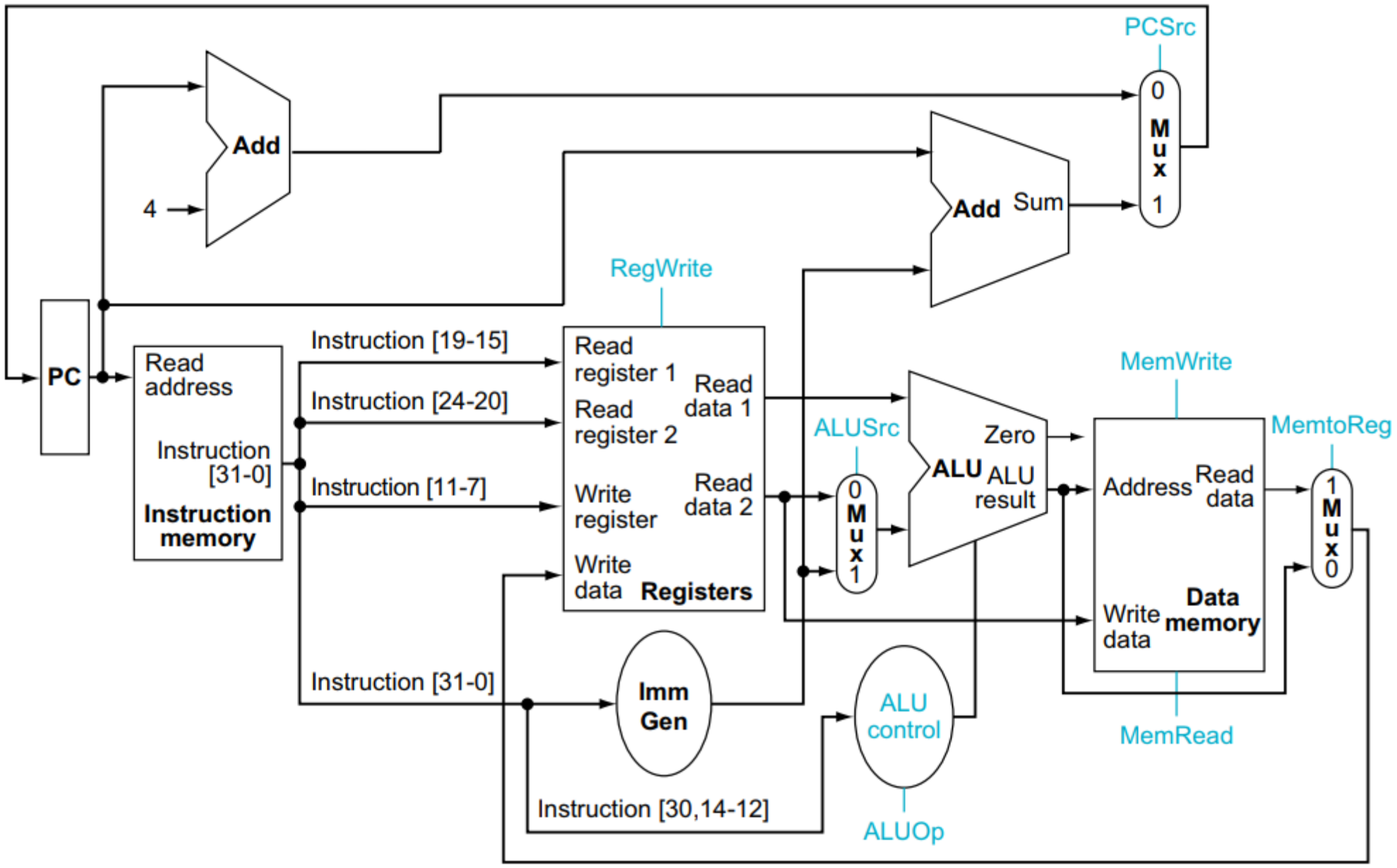
add ve **sub** ise func3 ve func7'den ayrışıyor

CPU CONTROL DESIGN

Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
lw	00	load word	XXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

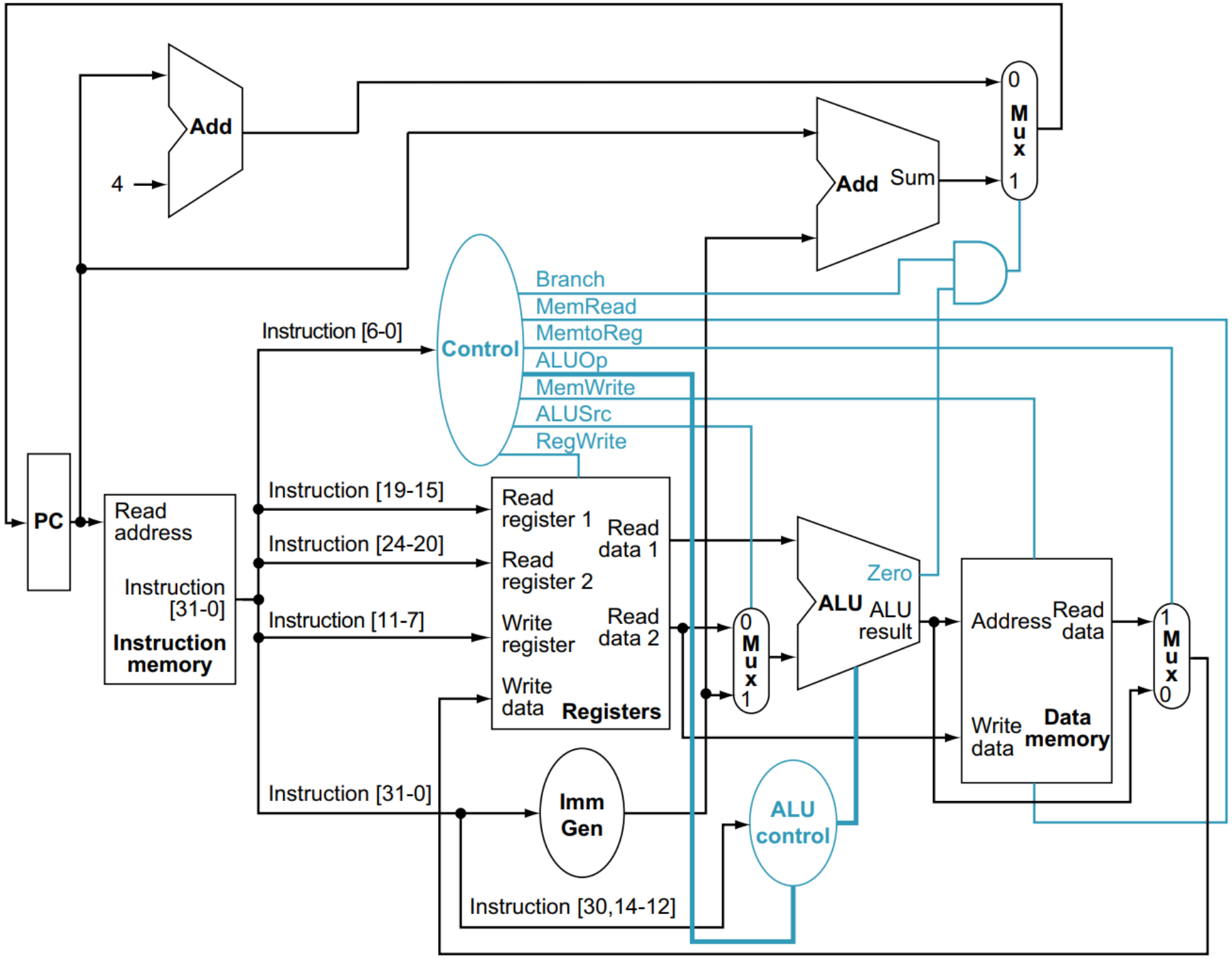
ALUOp		Funct7 field							Funct3 field			
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	Operation
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

CPU CONTROL & DATAPATH DESIGN



CPU CONTROL SIGNALS

Signal name	Effect when deasserted	Effect when asserted
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, 12 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.



CPU CONTROL SIGNALS

Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
lw	1	1	1	1	0	0	0	0
sw	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

Input or output	Signal name	R-format	lw	sw	beq
Inputs	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Outputs	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

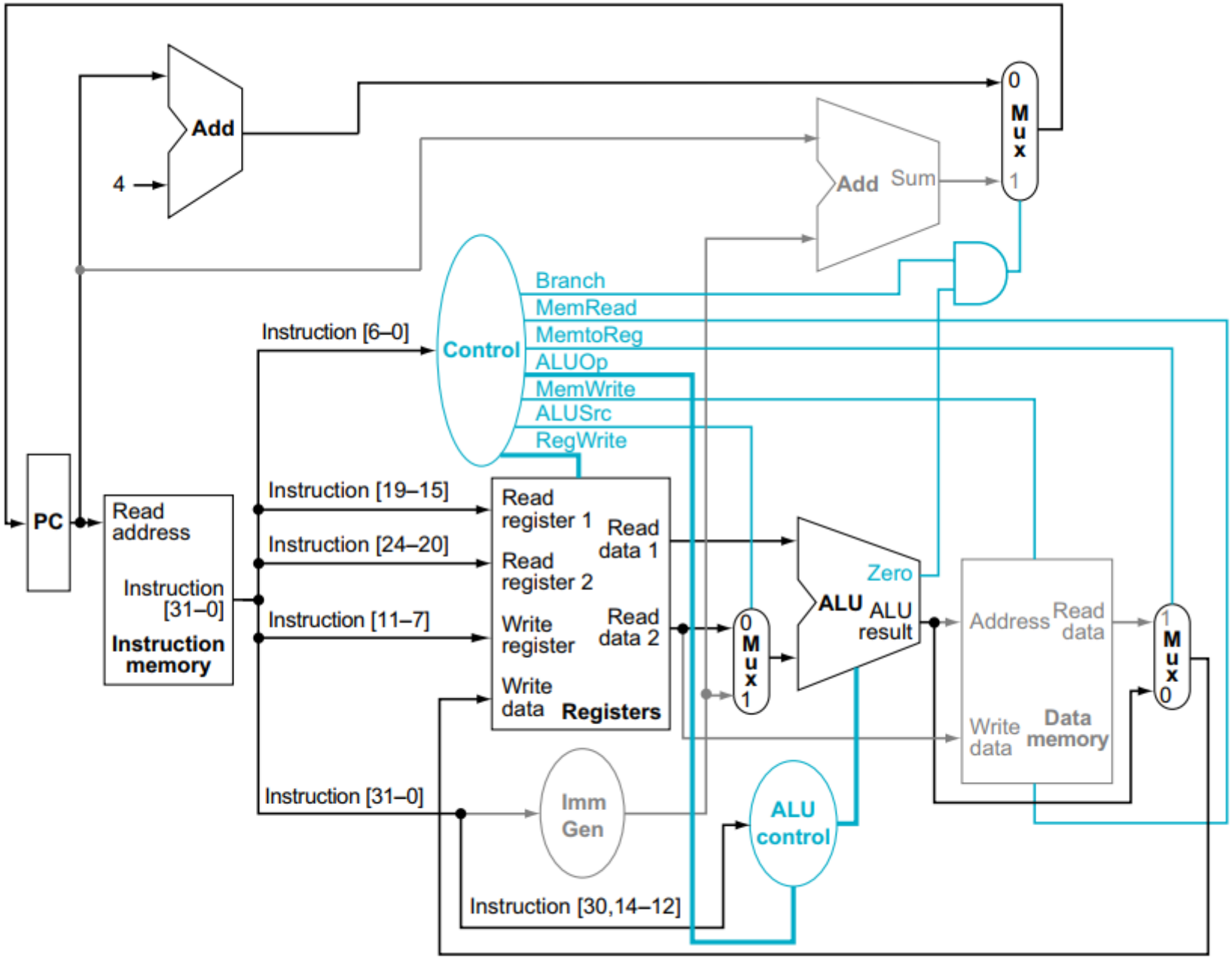


FIGURE 4.23 The datapath in operation for an R-type instruction, such as `add x1, x2, x3`. The control lines, datapath units, and connections that are active are highlighted.

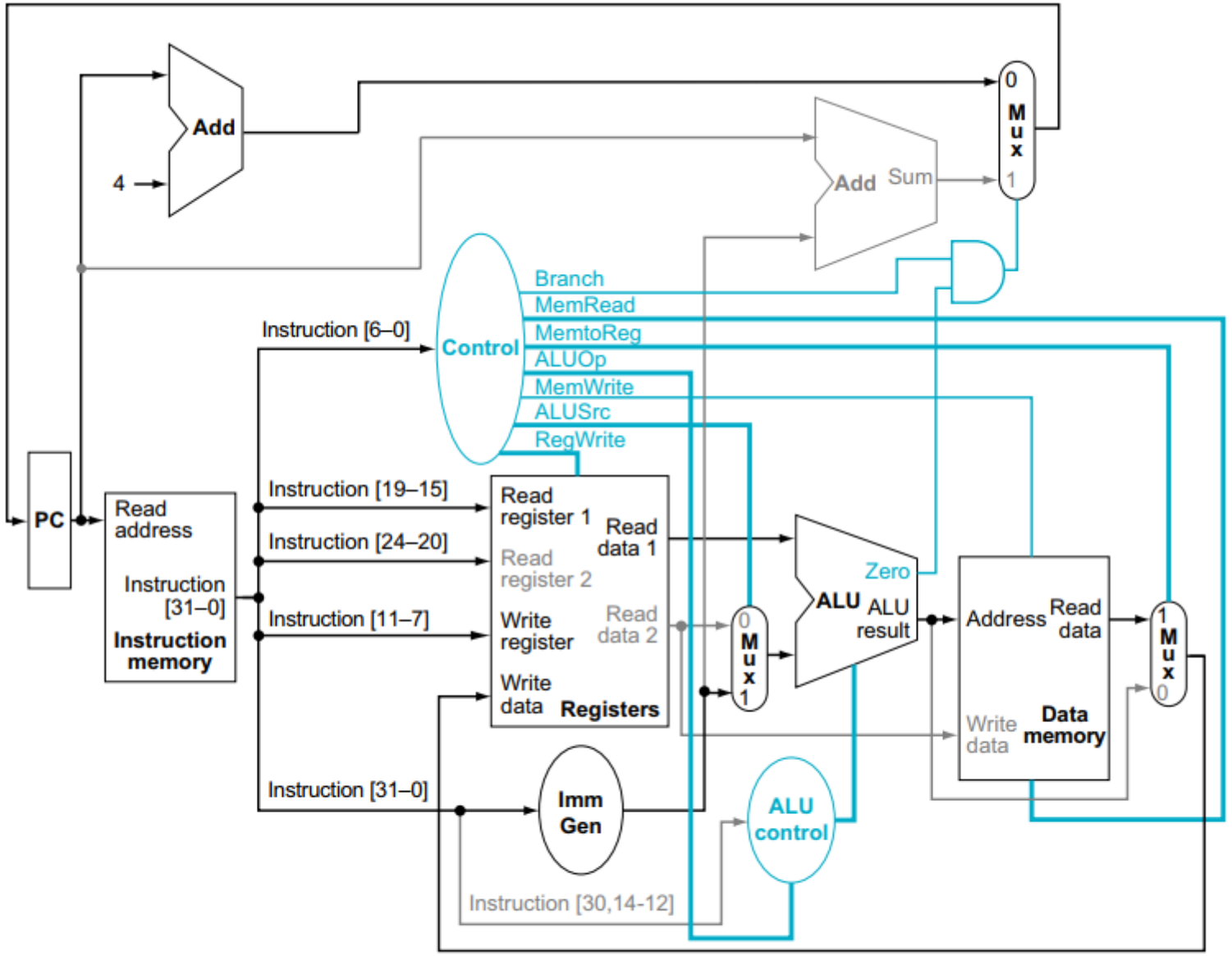


FIGURE 4.24 The datapath in operation for a load instruction. The control lines, datapath units, and connections that are active are highlighted. A store instruction would operate very similarly. The main difference would be that the memory control would indicate a write rather than a read, the second register value read would be used for the data to store, and the operation of writing the data memory value to the register file would not occur.

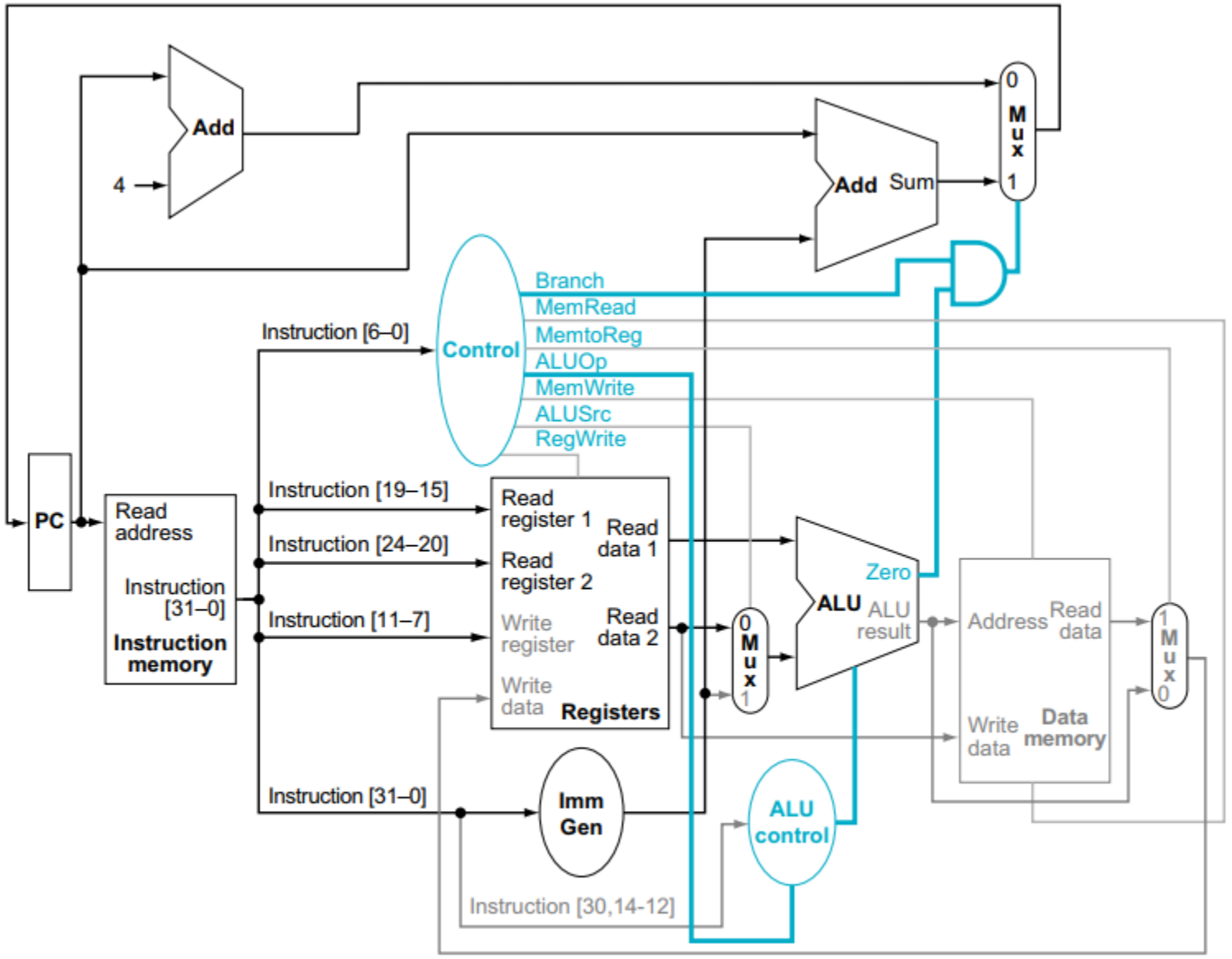


FIGURE 4.25 The datapath in operation for a branch-if-equal instruction. The control lines, datapath units, and connections that are active are highlighted. After using the register file and ALU to perform the compare, the Zero output is used to select the next program counter from between the two candidates.