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

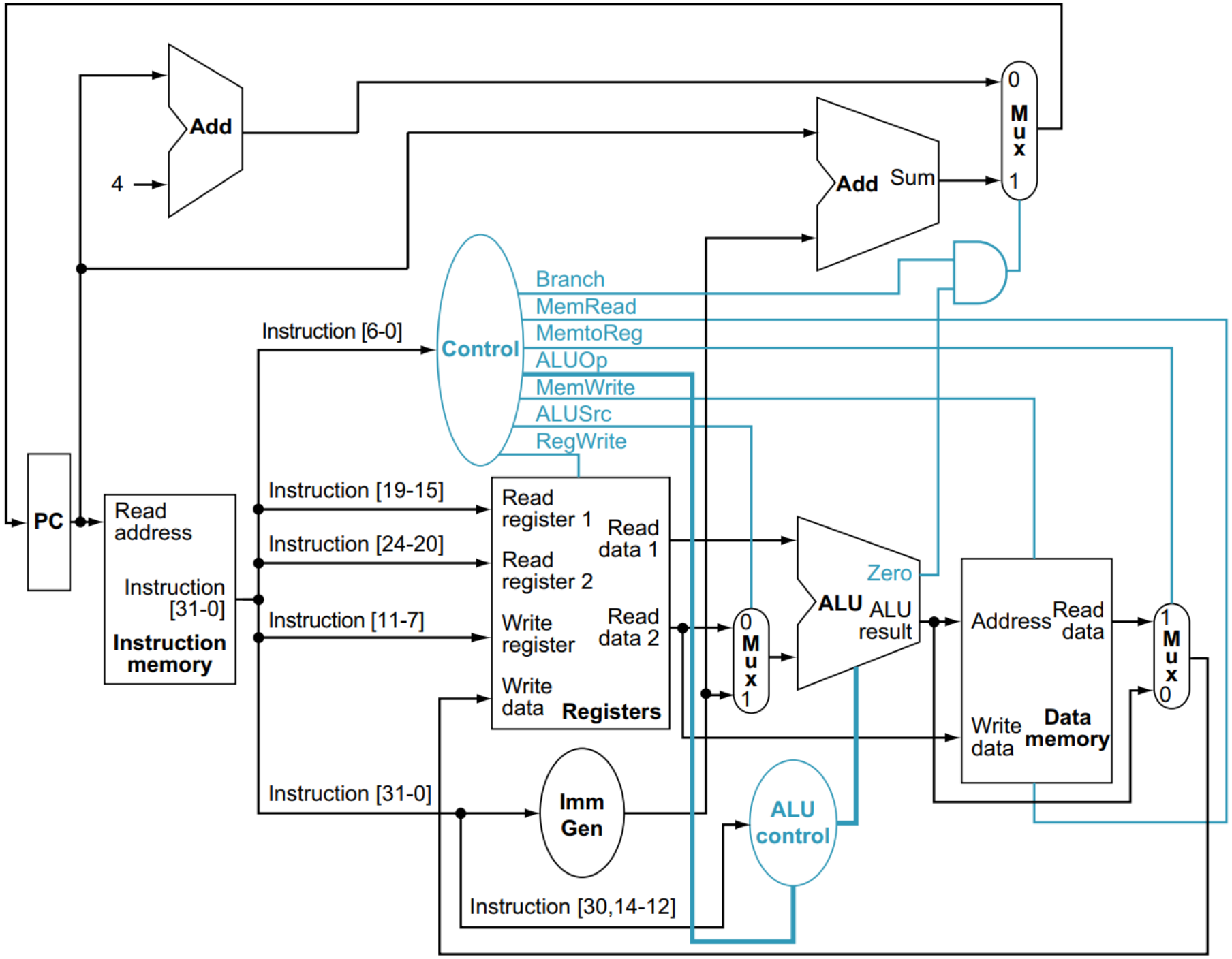
10. Hafta: Tek-çevrim işlemci tasarımı (single-cycle CPU)

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

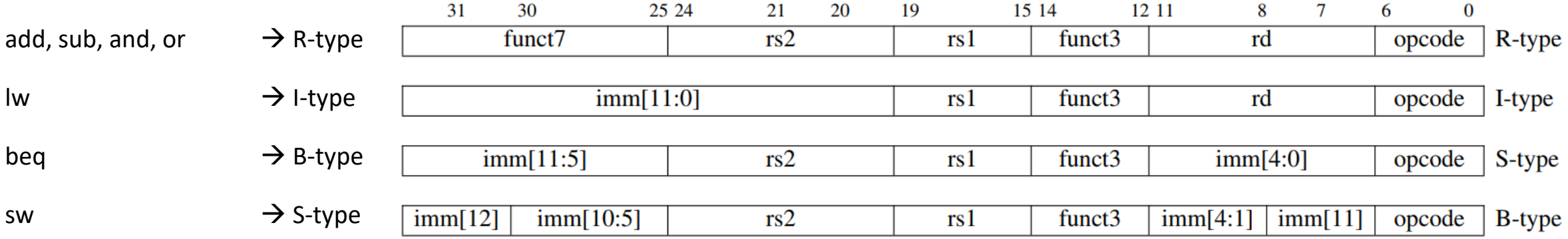


CPU CONTROL SIGNALS

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

SINGLE-CYCLE CPU PERFORMANCE



Instructionların veriyolunda izledikleri yol ve tahmini gecikmelerini hesaplamaya çalışalım: **R-Type →**

Clock rising edge sonrasında PC registerının değeri güncellenir: **PC FF clock-to-q delay**

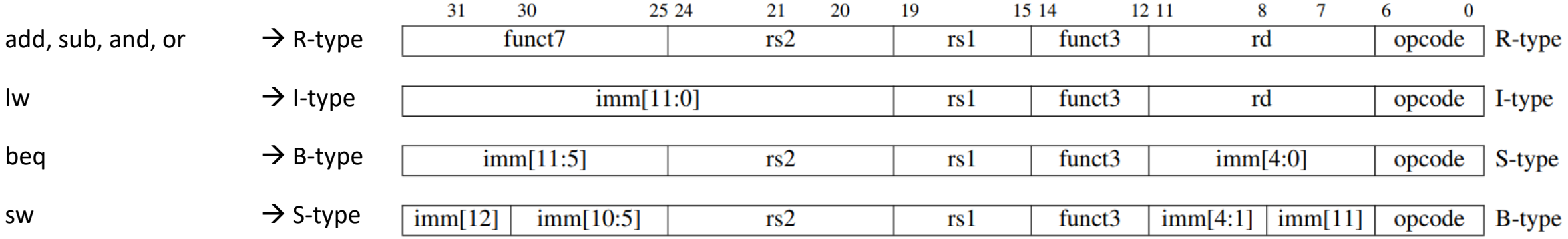
Güncel PC değeri instruction memory adres girişi olur, instruction memory'den instruction okunur: **Memory read**

Register file'dan 2 adet register okunur, 2. register ayrıca mux'tan geçer, bu sırada control logic'de mux için ALUsrc sinyalinin de hesaplanmış olması gerekir: **RegFile read + mux** (ALUsrc daha önce hesaplanır varsayımıyla)

ALU'da aritmetik ya da mantıksal işlem gerçekleştirilir: **ALU**

ALU sonucu MemtoReg control sinyali ile kontrol edilen mux'tan geçerek register file'a yazılır, yazma işlemi bir sonraki clock rise edge'de gerçekleşecektir, register file'a yazılacak verinin setup time öncesinde hazır olması gerekir: **RegFile setup time**

SINGLE-CYCLE CPU PERFORMANCE



Instructionların veriyolunda izledikleri yol ve tahmini gecikmelerini hesaplamaya çalışalım: **lw →**

Clock rising edge sonrasında PC registerının değeri güncellenir: **PC FF clock-to-q delay**

Güncel PC değeri instruction memory adres girişi olur, instruction memory'den instruction okunur: **Memory read**

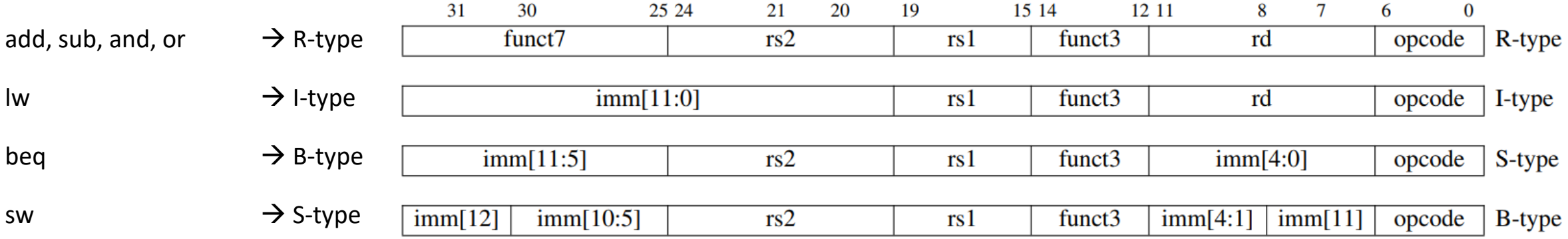
Register file'dan memory adres hesabı için 1 adet register okunur, adresin offset bilgisi immgen'den gelir ve mux'tan geçer, memory okumak imm hesaplamaktan daha uzundur: **MAX(RegFile read, immgen + mux) = RegFile read**

ALU'da toplama işlemi gerçekleştirilir: **ALU**

ALU sonucu data memory için adres olur, data memory'den okuma yapılır: **Memory read + mux**

gerçekleşecektir, register file'a yazılacak verinin setup time öncesinde hazır olması gerekir: **RegFile setup time**

SINGLE-CYCLE CPU PERFORMANCE



Instructionların veriyolunda izledikleri yol ve tahmini gecikmelerini hesaplamaya çalışalım: **beq →**

Clock rising edge sonrasında PC registerının değeri güncellenir: **PC FF clock-to-q delay**

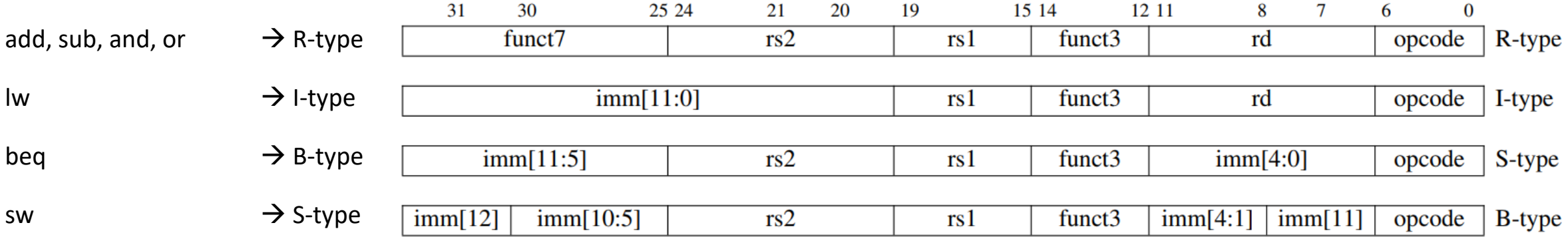
Güncel PC değeri instruction memory adres girişi olur, instruction memory'den instruction okunur: **Memory read**

Register file'dan eşitlik kontrolü için 2 adet register okunur, 2. register mux'tan geçer: **RegFile read + mux**

ALU'da çıkarma işlemi gerçekleştirilir: **ALU**

ALU sonucu zero sinyaline göre , PC'ye PC + IMM ya da PC+4 değeri yazılır. Yavaş olan yol: **IMM + ADD + mux**

SINGLE-CYCLE CPU PERFORMANCE



Instructionların veriyolunda izledikleri yol ve tahmini gecikmelerini hesaplamaya çalışalım: **sw →**

Clock rising edge sonrasında PC registerının değeri güncellenir: **PC FF clock-to-q delay**

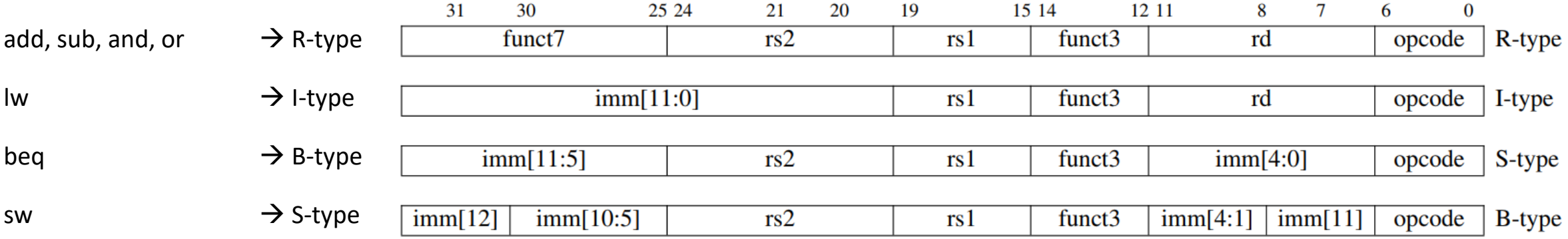
Güncel PC değeri instruction memory adres girişi olur, instruction memory'den instruction okunur: **Memory read**

Register file'dan 2 adet register okunur, 1 tanesi memory adres hesabı için kullanılır. Adresin offset bilgisi immgen'den gelir ve mux'tan geçer, memory okumak imm hesaplamaktan daha uzundur: **MAX(RegFile read, immgen + mux) = RegFile read**

ALU'da toplama işlemi gerçekleştirilir: **ALU**

ALU sonucu data memory için adres olur, data memory'ye diğer okunan register değeri yazılır: **Memory write**

SINGLE-CYCLE CPU PERFORMANCE



Bütün instructionlar için toplam gecikme hesapları:

R-Type → tcq_PC + Mem_Read + RF_Read + Mux + ALU + RF_Setup

lw → tcq_PC + Mem_Read + RF_Read + ALU + Mem_Read + Mux + RF_Setup

beq → tcq_PC + Mem_Read + RF_Read + Mux + ALU + Imm + Add + Mux

sw → tcq_PC + Mem_Read + RF_Read + ALU + Mem_Write

Single-cycle CPU'da bütün instructionlar tek bir saat vuruşunda, yani 1 periyot süresi boyunca işlemini tamamlamak durumundadır. Bu periyot, yani işlemci frekans değeri de en uzun gecikmeye sahip instruction'a göre hesaplanır. RISC mimarisinde load instructionları, en fazla aşamadan geçen ve en fazla gecikmeye sahip instructionlardır

SINGLE-CYCLE CPU PERFORMANCE

Bütün instructionlar için toplam gecikme hesapları:

R-Type → tcq_PC + Mem_Read + RF_Read + Mux + ALU + RF_Setup

lw → tcq_PC + Mem_Read + RF_Read + ALU + Mem_Read + Mux + RF_Setup

beq → tcq_PC + Mem_Read + RF_Read + Mux + ALU + Imm + Add + Mux

sw → tcq_PC + Mem_Read + RF_Read + ALU + Mem_Write

Mesela R-Type, beq veya sw instructionları da farklı gecikmeye sahiptirler. Fakat en uzun gecikme lw gecikmesi olduğu için bu instructionlar işlerini daha önce bitirseler dahi periyot boyunca beklemek durumundadırlar.

Single-cycle CPU'larda CPI (cycle-per-instrcution) değeri 1'dir. Performans denkleminde bu güzel bir şey olsa da yine aynı denklemdeki periyot bileşenini arttırdığı için Single-cycle CPU performansı düşüktür.

SINGLE-CYCLE CPU PERFORMANCE

Element	Parameter	Delay (ps)
Register clk-to-Q	t_{pcq}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	30
AND-OR gate	$t_{AND\text{-}OR}$	20
ALU	t_{ALU}	120
Decoder (control unit)	t_{dec}	25
Extend unit	t_{ext}	35
Memory read	t_{mem}	200
Register file read	t_{RFread}	100
Register file setup	$t_{RFsetup}$	60

lw → tcq_PC + Mem_Read + RF_Read + ALU + Mem_Read + Mux + RF_Setup

Min Periyot = 40 + 200 + 100 + 120 + 200 + 30 + 60 = 750 ps → Max Frequency = 1 / 750 ps = 1.33 GHz

SINGLE-CYCLE CPU PERFORMANCE

Element	Parameter	Delay (ps)
Register clk-to-Q	t_{pcq}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	30
AND-OR gate	$t_{AND\text{-}OR}$	20
ALU	t_{ALU}	120
Decoder (control unit)	t_{dec}	25
Extend unit	t_{ext}	35
Memory read	t_{mem}	200
Register file read	t_{RFread}	100
Register file setup	$t_{RFsetup}$	60

R-Type → tcq_PC + Mem_Read + RF_Read + Mux + ALU + RF_Setup | Min Periyot = 40 + 200 + 100 + 30 + 120 + 60 = 550 ps

beq → tcq_PC + Mem_Read + RF_Read + Mux + ALU + Imm + Add + Mux | Min Periyot = 40 + 200 + 100 + 30 + 120 + 35 + 90 + 30 = 645 ps

sw → tcq_PC + Mem_Read + RF_Read + ALU + Mem_Write | Min Periyot = 40 + 200 + 100 + 120 + 200 = 660 ps

lw → 750 ps

MULTI-CYCLE CPU

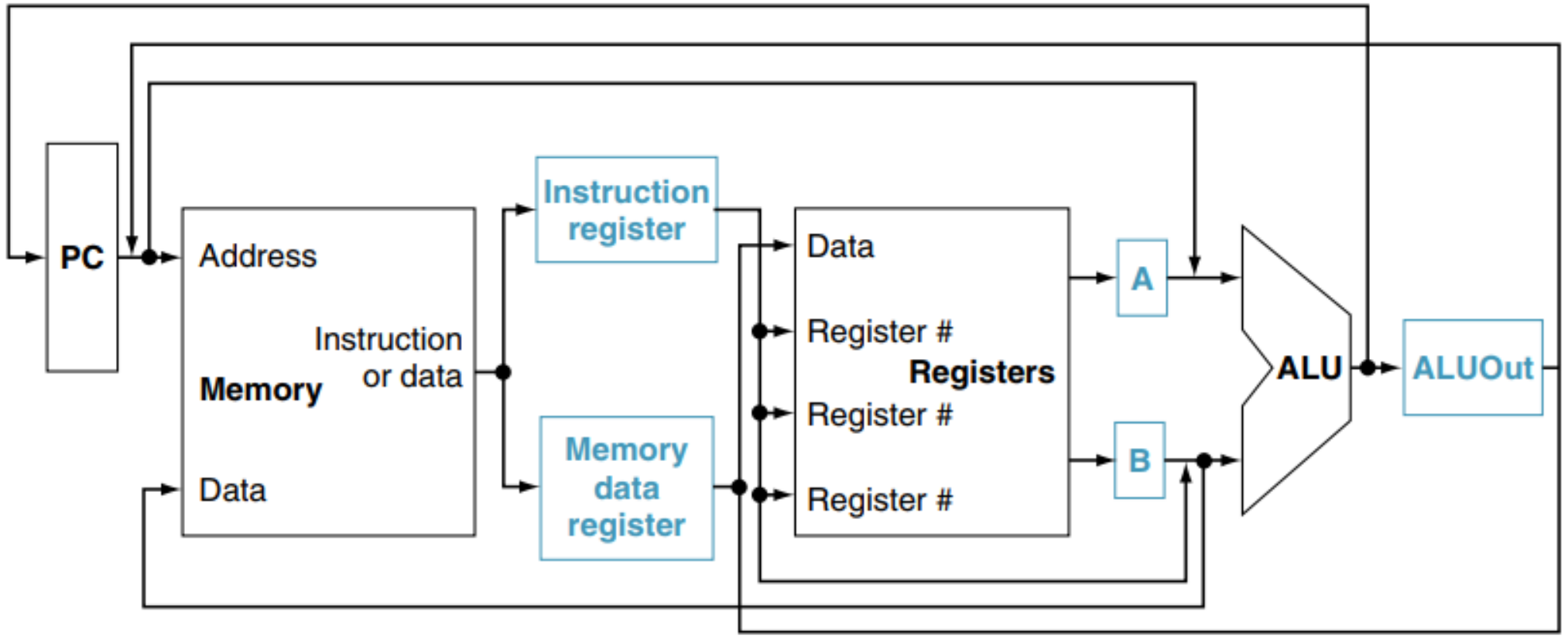


FIGURE e4.5.1 The high-level view of the multicycle datapath. This picture shows the key elements of the datapath: a shared memory unit, a single ALU shared among instructions, and the connections among these shared units. The use of shared functional units requires the addition or widening of multiplexors as well as new temporary registers that hold data between clock cycles of the same instruction. The additional registers are the Instruction register (IR), the Memory data register (MDR), A, B, and ALUOut.

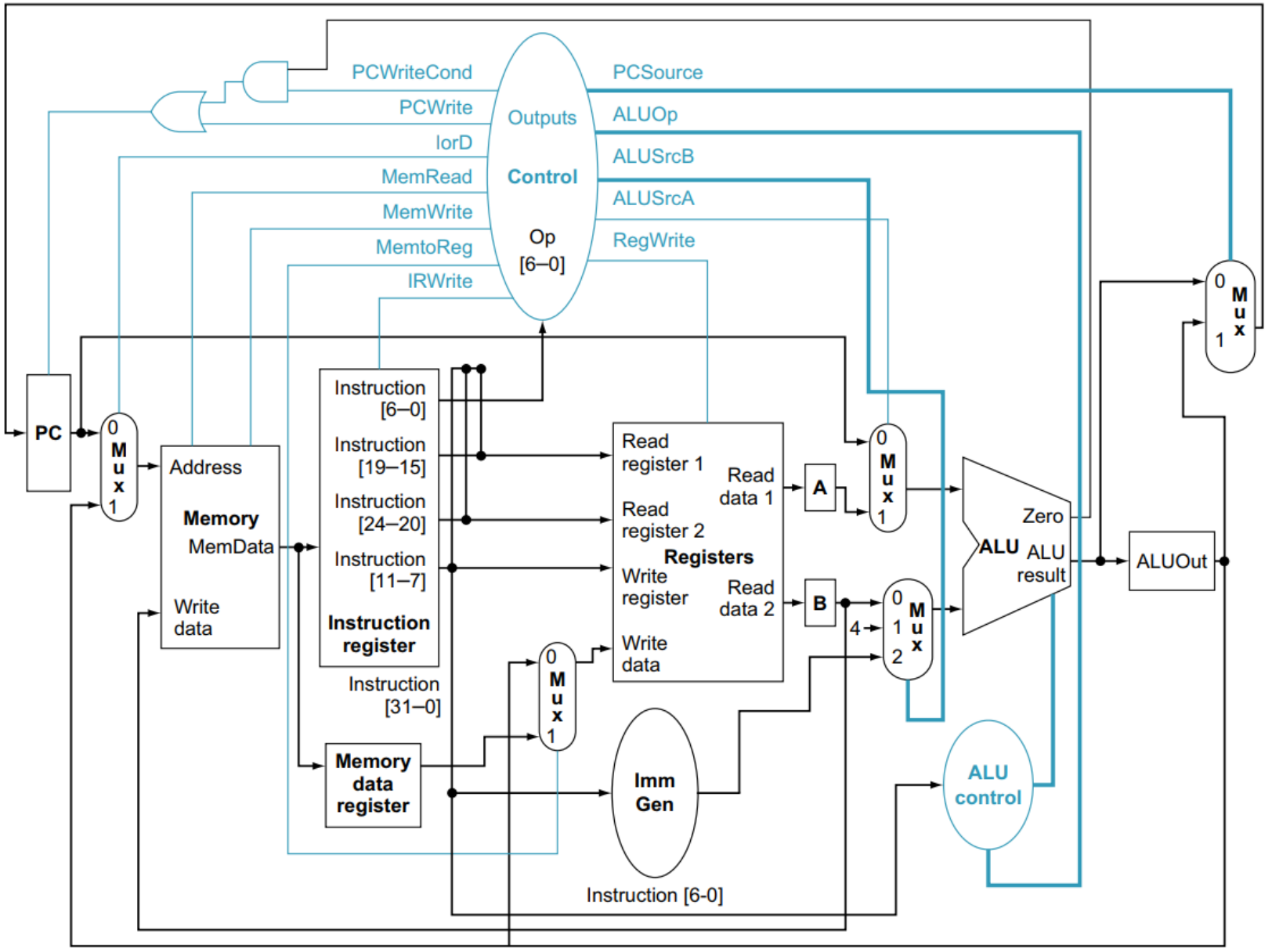


FIGURE e4.5.4 The complete datapath for the multicycle implementation together with the necessary control lines. The control lines of Figure e4.5.3 are attached to the control unit, and the control and datapath elements needed to effect changes to the PC are included. The major additions from Figure 4.29 include the multiplexor used to select the source of a new PC value; gates used to combine the PC write signals; and the control signals PCSource, PCWrite, and PCWriteCond. The PCWriteCond signal is used to decide whether a conditional branch should be taken.

MULTI-CYCLE CPU – EXECUTION STEPS

Her bir adım, 1 saat vuruşunda gerçekleşecek. RISC-V ISA'da instructionlar için CPI 3,4,5 olabilir.

1. Adım - Instruction Fetch: Bu aşamada instruction memory'den 32-bitlik instruction okunur ve bir sonraki sıradaki instruction için adres hesaplanır.

IR <= Memory[PC];

PC <= PC + 4;

2. Adım – Instruction Decode and Register Fetch: 1. adımda hangi instruction olduğunu henüz bilmiyoruz, bunun anlaşılabilmesi için instruction fieldleri (opcode, func3, func7) çözülmelidir ve yine bu aşamada register file'dan hangi registerların okunacağı adresi belirlenir. Ayrıca instruction'ın branch olma ihtimaline göre ALUout register için PC'ye offset değeri eklenir.

A <= Reg[IR[19:15]];

B <= Reg[IR[24:20]];

ALUOut <= PC + immediate;

MULTI-CYCLE CPU – EXECUTION STEPS

3. Adım - Execution, memory address computation, or branch completion: Bu aşamaya kadar instruction'ın ne olduğundan bağımsız işlemler gerçekleştirildi. Bu aşamada ise instruction'a göre işlemler devam edecek. Örneğin lw veya sw instructionları için ALU'da bir registerla immediate değeri toplanacak ve ALUout registerına bu sonuç yazılacak. Eğer R-Type bir instruction ise, A ve B registerlarındaki değerler instruction tipine göre işlenecek. Eğer beq instruction ise ALU'da çıkarma işlemi yapılacak ve zero kontrol sinyali hesaplanmış olunacak ve PC update edilecek.

if (A == B) PC <= ALUOut;

Burada dikkat edilmesi gereken şey, ALUOut registerında bir önceki saat vuruşunda (2. Adım) PC + immediate değerinin yazılmış olmasıdır. Eğer fetch edilen instruction bir branch instruction ise bu adımda bu instruction tamamlanmış olur ve bir sonraki saat vuruşunda yeni instruction çekilebilir.

MULTI-CYCLE CPU – EXECUTION STEPS

4. Adım - Memory access or R-type instruction completion step: Bu aşamada lw veya sw instruction memory erişimi sağlar, instructiona göre yazma ya da okuma yapar. Eğer R-Type bir instruction ise bu instruction tamamlanır. Memory'den veri getiriliyorsa (lw) getirilen veri MDR registerına kaydedilir ki bir sonraki saat vuruşunda kullanılabilsin. Memory'ye veri yazılıyorsa da ALUout registerına bir önceki adımda (3. Adım) hesaplanan adrese B registerının içeriği yazılır.

MDR <= Memory [ALUOut];

Memory [ALUOut] <= B;

R-Type instructionlar içinse bir önceki saat vuruşunda (3. Adım) hesaplanmış ve ALUout registerına yazılmış olan ALU sonucu bu adımda register file'da ilgili adrese yazılır.

Reg[IR[11:7]] <= ALUOut;

Bu işlem ile de 4. saat vuruşu sonunda R-Type instructionlar veya sw tamamlanmış olur.

MULTI-CYCLE CPU – EXECUTION STEPS

5. Adım - Memory read completion step: Bu aşamaya sadece lw instructionı gelebilir, diğer instructionlar işlemini tamamlamış olurlar. lw instructionı ise bu aşamada son olarak memory'den okunan ve MDR registerında tutulan değeri register file'a yazar.

Reg[IR[11:7]] <= MDR;

Bu işlem ile de 5. saat vuruşu sonunda lw instruction tamamlanmış olur.

MULTI-CYCLE CPU – EXECUTION STEPS

Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches
Instruction fetch	IR <= Memory[PC] PC <= PC + 4		
Instruction decode/register fetch	A<= Reg [IR[19:15]] B <= Reg [IR[24:20]] ALUOut <= PC + immediate		
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + immediate	if (A == B) PC <= ALUOut
Memory access or R-type completion	Reg [IR[11:7]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B	
Memory read completion		Load: Reg[IR[11:7]] <= MDR	

FIGURE e4.5.6 Summary of the steps taken to execute any instruction class. Instructions take from three to five execution steps. The first two steps are independent of the instruction class. After these steps, an instruction takes from one to three more cycles to complete, depending on the instruction class. The empty entries for the Memory access step or the Memory read completion step indicate that the particular instruction class takes fewer cycles. In a multicycle implementation, a new instruction will be started as soon as the current instruction completes, so these cycles are not idle or wasted. As mentioned earlier, the register file actually reads every cycle, but as long as the IR does not change, the values read from the register file are identical. In particular, the value read into register B during the Instruction decode stage, for a branch or R-type instruction, is the same as the value stored into B during the Execution stage and then used in the Memory access stage for a store word instruction.

MULTI-CYCLE CPU – CPI CALCULATION

SORU: Multi-Cycle bir işlemcide bir program derlendiğinde %20 load, %8 store, %10 branch ve geri kalan %62 ALU (R-Type) instructionlardan oluşuyor. Bu instructionların saat çevrim sayıları sırasıyla 5,4,3 ve 4 ise bu program için bu işlemcide CPI değeri nedir?

CEVAP:

0.2*5 + 0.08*4 + 0.1*3 + 0.62*4 = 4.10