5.5 MULTITAKTNA IMPLEMENTACIJA

Jednotaktna arhitektura predvidja izvršavanje instrukcija obavljenjem niza koraka, navedenih u tabeli 2 u poglavlju 5.2, koji odgovaraju funkcionisanju upotrijebljenih elemenata. Prilikom implementacije svake od instrukcija, ovi koraći se obavljaju jedan za drugim i svi oni zajedno se izvršavaju u toku trajanja jednog taktnog intervala. Posljednja osobina implicira multipliciranje svih funkcionalnih elemenata koje je potrebno upotrijebiti više nego jedan put za vrijeme izvršavanja instrukcije, što je detaljno elaborirano u sekciji 5.4.

Ipak, sve implementirane instrukcije ne zahtijevaju obavljanje istih koraka, niti, što je još važnije, istog broja koraka tokom svog izvršavanja. Time se kod jednotaktne implementacije implicira i potencijalno rasipanje značajnog vremena prilikom izvršavanja većine istrukcija. Naime, dužina taktnog intervala mora biti odredjena tako da obezbijedi pouzdano izvršavanje svih instrukcija, pa i vremenski najzahtjevnije instrukcije (*lw*, koja zahtijeva obavljanje 5 koraka tokom svog izvršavanja – pogledaj tabelu 2). Medjutim, pošto je taktni interval fiksne dužine, kod jednotaktne implementacije je i izvršavanje ostalih instrukcija takodje odredjeno istim taktnim intervalom. Drugim riječima, prilikom implementacije instrukcija uslovnog i bezuslovnog skoka, koje zahtijevaju 3 koraka za svoje izvršavanje, rasipa se približno 40% vremena (odnosno, rasipaju se 2 nepotrebna od ukupno 5 koraka sa kojima je odredjena fiksna dužina taktnog intervala). Sa druge strane, prilikom implementacije instrukcija koje zahtijevaju 4 koraka za svoje izvršavanje, rasipa se približno 20% vremena (odnosno, rasipa se 1 nepotrebni od ukupno 5 koraka sa kojima je odredjena fiksna dužina taktnog intervala).

U cilju prevazilaženja navedenih nedostataka jednotaktne arhitekture, implementira se multitaktna (višetaktna, odnosno eng. *multiple clock-cycle*) arhitektura. Multitaktna arhitektura predvidja izvršavanje instrukcija obavljenjem istog niza koraka, kao u slučaju jednotaktnog dizajna, navedenih u tabeli 2, ali za razliku od jednotaktne arhitekture, svaki od koraka obavlja se u toku posebnog taktnog intervala. Na ovaj način, postižu se sljedeće karakteristike:

- Svaka instrukcija uzima odredjeni broj taktnih intervala za svoje izvršavanje, i to onaj broj taktova koji odgovara koracima koje instrukcija zahtijeva. Preciznije, instrukcije uslovnog i bezuslovnog skoka, koje zahtijevaju 3 koraka, biće izvršavane u toku 3 taktna intervala, instrukcija *lw*, koja zahtijeva 5 koraka, biće izvršavana u toku 5 taktnih intervala, a instrukcije koje zahtijevaju 4 koraka, biće izvršavane u toku 4 taktna intervala,
- Ne multipliciraju se funkcionalni elementi koji se upotrebljavaju u implementaciji, čime se redukuje hardware-ska složenost multitaktne implementacije u poredjenju sa jednotaktnom. Naime, u jednotaktnoj implementaciji, slika 8, ALU se multiplicira 3 pita (ALU i 2 sabirača), a memorija 2 puta (Instruction i Data memory). Kod multitaktne implementacije, za izvršavanje instrukcija upotrebljavaju se najmanje 3 taktna intervala, tako da se bilo koji elemenat, pa i ALU, može najmanje 3 puta upotrijebiti tokom izvršavanja bilo koje instrukcije (ograničenje je da se elementi mogu upotrijebiti tačno jedan put po taktnom intervalu),
- Svaka instrukcija upotrebljava samo neophodan broj taktnih intervala za svoje izvršavanje i nema rasipanja vremena u smislu da neka instrukcija, ili više njih uzimaju više taktnih intervala nego što je neophodno. U tom smislu, na nivou više izvršavanih instrukcija (djelova programa ili čitavih programa), multitaktna implementacija može poboljšati vrijeme izvršavanja zahtijevano jednotaktnom implementacijom.

  NAPOMENA 1: Dužina taktnog intervala multitaktne implementacije treba da obezbijedi pouzdano izvršavanje svakog od koraka koji se obavlja tokom izvršavanja instrukcija (pogledaj tabelu 2). Shodno tome, dužina ovog taktnog intervala odredjena je vremenski najzahtjevnijim korakom. Primijetimo da pristup memoriji računara (u cilju čitanja ili upisivanja podataka) zahtijeva najduže vrijeme za svoje izvršavanje, tako da je izvršavanjem ove akcije odredjena dužina taktnog intervala multitaktne implementacije. Ipak, pošto je taktni interval fiksne dužine, sve ostale akcije/koraci, koje zahtijevaju kraće vrijeme za svoje izvršavanje od pristupa memoriji, izvršavaju se u toku istog taktnog intervala, što takodje dovodi do izvjesnog rasipanja vremena. Sada je neophodno procijeniti: u kom slučaju

(jednotaktne ili multitaktne implementacije) je rasipanje vremena značajnije i, shodno tome, koja implementacija obezbjedjuje bolje vremenske karakteristike?

Odgovor na ovo pitanje može se dati sa 2 nivoa:

- Izvršavanja jedne intrukcije – na primjer vremenski najzahtjevnije *lw*, ili
- Izvršavanja grupe instrukcija.

NAPOMENA 2: Odgovor posmatran sa nivoa jedne instrukcije prilično je jednostavan. Jednotaktnom implementacijom ne rasipa se vrijeme prilikom implementacije instrukcije *lw*. Naime, taktni interval jednotaktne implementacije odredjen je vremenom neophodnim za izvršavanje ove instrukcije, a koraci neophodni za njeno izvršavanje uzimaju tačno onoliko vremena koliko je potrebno za njihovo obavljanje. Sa druge strane, kod multitaktne implementacije instrukcije *lw*, svi koraci koji zahtijevaju kraće vrijeme od vremena pristupa memoriji (memoriji se pristupa u I i u IV koraku izvršavanja *lw* instrukcije) uzrokuju rasipanje vremena. Drugim riječima, jednotaktnom implementacijom se optimizira vrijeme izvršavanja pojedinačne insttrukcije *lw*.

Odgovor na postavljeno pitanje s aspekta grupe izvršavanih instrukcija (dijela ili čitavog programa) značajno je složeniji i zahtijeva dublju analizu. U cilju njegovog kreiranja, posmatrajmo sljedeći primjer izvršavanja grupe instrukcija.

Primjer: Pretpostavimo da se izvršava grupa od 5 instrukcija, od kojih je jedna *lw* instrukcija, jedna instrukcija uslovnog ili bezuslovnog skoka, koja zahtijeva 3 koraka za svoje izvršavanje, te 3 instrukcije koje zahtijevaju 4 koraka za svoja izvršavanja. Pretpostavimo da je za pristup memoriji potrebno vrijeme od 12 jedinica vremena (j.v.), za izvršavanje operacije ALU – 9 j.v., a za upisivanje podataka u Registers jedinicu 8 j.v. Napomenimo da pretpostavljene proporcije u vremenima izvršavanja prethodno navedenih akcija odgovaraju realnoj situaciji.

Izračunajmo najprije zahtijevanu dužinu taktnog intervala kod jednotaktne implementacije (T_{SCI}, gdje skraćenica SCI odgovara Single Clock-cycle Implementation – jednotaktnoj implementaciji), a potom i kod multitaktne implementacije (T_{MCI}, gdje skraćenica MCI odgovara Multiple Clock-cycle Implementation – multitaktnoj implementaciji):

- Kod jednotaktne implementacije, dužina taktnog intervala odredjena je vremenom neophodnim za izvršavanje vremenski najzahtjevnije instrukcije *lw*. Razmatranjem tabele 2 iz sekcije 5.2, možemo primijetiti da *lw* instrukcije zahtijeva
  - dvostruki pristup memoriji (u I koraku za čitanje instrukcije iz memorije i u IV koraku za čitanje podatka iz memorije),
  - izračunavanje adrese lokacije (od strane ALU) sa koje će podatak biti pročitan (u III koraku),
  - dekodiranje instrukcije (u II koraku) koje se izvršava u paraleli sa izračunavanjem (od strane sabirača/ALU) ciljne adrese grananja Target, tako da će vrijeme izvršavanja ovog koraka biti odredjeno vremenom potrebnim sabiraču/ALU za izračunavanje ciljne adrese grananja,
  - upisivanje podatka u odredjeni registar Registers jedinice (V korak).

  Shodno prethodno navedenom, te pretpostavljenim vremenskih zahtjevima pojedinih operacija, zaključujemo da dužina taktnog intervala kod jednotaktne implementacije iznosi:

  $$T_{SCI}=(2\times12+2\times9+8)\text{ j.v.}=50\text{ j.v.},$$

  te da je T_{SCI} ujedno vrijeme izvršavanja svih implementiranih instrukcija.
- Kod multitaktne implementacije, dužina taktnog intervala odredjena je vremenom neophodnim za izvršavanje vremenski najzahtjevnije operacije/koraka koji se obavlja tokom izvršavanja instrukcija. Već je navedeno da je pristup memoriji vremenski najzahtjevija operacija, te da se ona obavlja u I koraku svih razmatranih instrukcija (pogledaj tabelu 2). Uz

to, pretpostavljeno je da da vrijeme pristupa memoriji iznosi 12 j.v., tako da dužina taktnog intervala kod multitaktne implementacije iznosi:

$$T_{MCI}=12 \text{ j.v.}$$

Vrijeme izvršavanja svake pojedinačne instrukcije kod multitaktne implementacija odgovara umnošku T_{MCI} sa brojem koraka koje ta instrukcija zahtijeva za svoje izvršavanje.

Vremena izvršavanja pojedinih instrukcija i ukupno vrijeme izvršavanja zadate grupe instrukcija u jednotaktnoj (SCI) i multitaktnoj (MCI) imaplementaciji predstavljeni su u tabeli 10.

Tabela 10. Grupa instrukcija zadatih u primjeru, vrijeme izvršavanja svake od njih u slučaju jednotaktne (SCI) i multitaktne implementacije (MCI), kao i ukupno vrijeme izvršavanja zadate grupe instrukcija.

Razmatrane instrukcije	SCI	MCI
(Bez)uslovni skok (3 CLK)	50 j.v.	36 j.v.
Instrukcija (4 CLK)	50 j.v.	48 j.v.
Instrukcija (4 CLK)	50 j.v.	48 j.v.
Instrukcija (4 CLK)	50 j.v.	48 j.v.
lw (5 CLK)	50 j.v.	60 j.v.
UKUPNO:	250 j.v.	240 j.v.

U skladu sa razmatranjima iz napomene 2 iz ovog poglavlja, primijetimo da jednotaktna imlementacija (SCI) poboljšava vrijeme izvršavanja instrukcije *lw* i to 20% u odnosu na multitaktnu implementaciju (MCI). Medjutim, multitaktna implementacija obezbjedjuje brže izvršavanje pretpostavljene grupe od 5 instrukcija. Preciznije, zavisno od instrukcija koje sačinjavaju grupu, multitaktna implementacija može obezbjedjivati poboljšane vremenske karakteristike u odnosu na odgovarajuću jednotaktnu implementaciju, i to iz razloga uzimanja više od jednog taktnog intervala po instrukciji i, što je još važnije, različitog (samo neophodnog) broja taktnih intervala za izvršavanje svake od različitih instrukcija pojedinačno. Drugim riječima, vrijeme izvršavanja ne mora biti nedostatak multitaktne implementacije u poredjenju sa odgovarajućom jednotaktnom.

ZAKLJUČAK: Saglasno navedenom, u odnosu na odgovarajuću jednotaktnu implementaciju, multitaktna implementacija

1. Obezbjedjuje značajne redukcije u upotrijebljenom hardware-u,
2. Zavisno od grupe instrukcija čije izvršavanje se zahtijeva, ne mora imati inferiorne vremenske karakteristike.

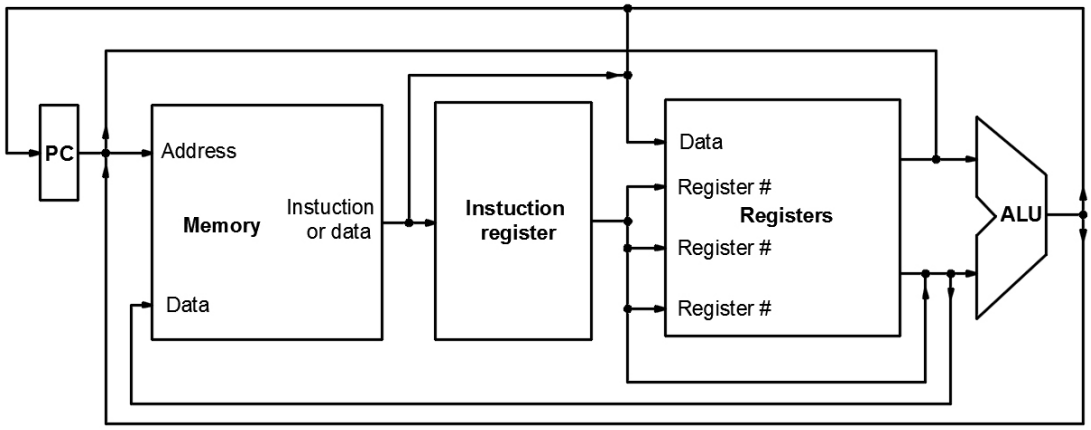
Shodno tome, multitaktna implementacija predstavlja praktično aplikabilan dizajn koji se upotrebljava u svim modernim računarskim sistemima. Stoga će joj u nastavku biti posvećena potpuna pažnja.

5.5.1 Datapath za multitaktnu implementaciju

U prethodnom poglavlju uočeno je da multitaktna arhitektura treba da obezbjedi implementaciju instrukcija iz posmatranog seta (instrukcije R-tipa (*add*, *sub*, *and*, *or*, *slt*), *lw*, *sw*, *beq* i *j*) u 3 do 5 taktnih intervala, odnosno upotrebu svake od zahtijevanih funkcionalnih jedinica 3 do 5 puta tokom izvršavanja svake instrukcije. Stoga je, u poredjenju sa jednotaktnom arhitekturom sa slike 8, kod multitaktne arhitekture nepotrebna upotrijeba ALU i 2 dodatna sabirača (2 "zaglupljene" ALU sa kontrolnim signalima 010), kao i dvije memorije (Instruction memory i Data memory), već samo

- jedne ALU koja će obavljati funkciju i ALU, sa slike 8, i 2 dodatna sabirača (u prva 3 koraka izvršavanja),
- jedne Memory koja će obavljati funkcije Instruction i Data memory sa slike 8.

Shodno navedenom, opšti pogled na multitaktni datapath za implementaciju razmatranog seta instrukcija (instrukcije R-tipa (*add*, *sub*, *and*, *or*, *slt*), *lw*, *sw*, *beq* i *j*) prikazan je na slici 9. Primijetimo da datapath sa slike 9, za razliku od jednotaktnog datapath-a sa slike 1, odnosno jednotaktnog dizajna



Slika 9. Sažeti prikaz datapath-a koji je namijenjen multitaktnom izvršavanju instrukcija.

sa slike 8, uključuje jednu memoriju i jednu ALU, ali i da su, shodno tome, dodatno share-ovani pojedini ulazi ovih jedinica, kao i da je dodat registar označen sa Instruction register (IR).

ZAPAŽANJE 1: ALU upotrijebljana kod multitaktnog datapath-a sa slike 9, obavlja funkcije ALU i dva dodatna sabirača upotrijebljenih kod jednotaktne arhitekture sa slike 8. Shodno tome, broj potencijalnih operanada ALU mora biti povećan, odnosno oba ulaza ALU sa slike 9 moraju biti share-ovana (svaki od njih za odgovarajuće ulaze ALU i ulaze 2 sabirača sa slike 8) u skladu sa razmatranjima prezentiranim u tabeli 11.

Tabela 11. Funkcionalni elementi/uredjaji (i njihovi ulazi) upotrijebljeni u jednotaktnoj arhitekturi sa slike 8 čije funkcije u miltitaktnoj implementaciji izvršava ALU sa slike 9.

Uredjaj	I ulaz	II ulaz
ALU	Reg(rs)	Reg(rt), sign_extend(address)
Add (za izračunavanja PC+4)	PC	Const. 4
Add (za izračunavanje Target)	PC+4	sign_extend(address)<<2

Kao i ranije, share-ovanje se vrši dodavanjem multipleksora sa odgovarajućim brojem ulaza na prvom i na drugom ulazu ALU sa slike 9. Primijetimo da će multipleksor na prvom ulazu biti novouveden u poredjenju sa jednotaktnom arhitekturom sa slike 8, a multipleksor na drugom ulazu ALU proširen u odnosu na odgovarajući multipleksor sa slike 8.

- Multipleksor na prvom ulazu ALU treba da ima 2 ulaza (za operande Reg(rs) i sadržaj PC registra, tabela 11), te jedan selekcioni ulaz, koji će biti nazvan shodno funkciji koju obavlja, odnosno *ALUSelA* (selektuje prvi, odnosno operand A ALU).

  NAPOMENA 1: Na prvi pogled, na osnovu razmatranja iz tabele 11, multipleksor na prvom ulazu ALU trebao bi da ima 3 ulaza (za operande Reg(rs), PC i PC+4). Medjutim, kao što je može primijetiti iz tabele 2 (poglavlje 5.2) i kao što će biti implementirano u nastavku, nakon izračunavanja (u prvom koraku/taktnom intervalu), PC+4 upisuje se nazad u PC registar, tako da u trenutku upotrebe PC+4 (u drugom koraku u cilju izračunavanja ciljne adrese Target), sadržaj PC registra suštinski je PC+4 s aspekta instrukcije koja se izvršava. Iz ovog razloga, multipleksor na prvom ulazu ALU treba da posjeduje 2 ulaza (za operande Reg(rs) i PC), gdje PC u prvom koraku sadrži adresu izvršavane instrukcije, dok u drugom koraku sadrži adresu instrukcije koja je u memoriji zapisana odmah nakon izvršavane instrukcije.

- Multipleksor na drugom ulazu ALU treba da ima 4 ulaza:
  - 2 ranije prepoznata ulaza (kod jednotaktne arhitekture sa slike 8) za operande Reg(rt) i sign_extend(address),
  - 1 ulaz za operand Const. 4, koja se dovodi na drugi ulaz prvog sabirača upotrijebljavanog kod jednotaktne arhitekture (slika 8) za formiranje adrese prve sljedeće lokacije (PC+4),

- 1 ulaz za operand sign_extend(address)<<2, koji se dovodi na drugi ulaz drugog sabirača upotrebljavanog kod jednotaktne arhitekture (slika 8) za formiranje ciljne adrese grananja Target.

Multipleksor na drugom ulazu ALU pored 4 ulaza za operande treba da sadrži i 2 selekciona ulaza, koji će biti nazvani shodno funkciji koju obavljaju, odnosno *ALUSelB* (selektuje drugi, odnosno operand B ALU).

ZAPAŽANJE 2: Memory upotrijebljana kod multitaktnog datapath-a sa slike 9, obavlja funkcije Instruction memory i Data memory upotrijebljenih kod jednotaktne arhitekture sa slike 8, u skladu sa razmatranjima prezentiranim u tabeli 12.

Tabela 12. Memorijski elementi/uredjaji (i njihovi adresni ulazi) upotrijebljeni u jednotaktnoj arhitekturi sa slike 8 čije funkcije u multitaktnoj implementaciji izvršava Memory jedinica sa slike 9.

Uredjaj	Read address	Write address
Instruction memory	PC	×
Data memory	ALUOut	ALUOut

Shodno činjenici, prikazanoj i u tabeli 12, da se obje ove memorije u jednotaktnoj implementaciji upotrebljavaju za uzimanje/čitanje instrukcija/podataka (Data memory i za upisivanje podataka), na Read address ulaz jedinstvene Memory (na slici 9, zbog njene opštosti i ne ulaženja u detalje, označen sa Address) može biti dovedena adresa sadržana u PC registru (u cilju uzimanja/čitanja instrukcije sa odgovarajuće lokacije), a može biti dovedena adresa izračunata od strane ALU (u cilju čitanja podatka sa odgovarajuće lokacije prilikom implementacije instrukcije *lw*). U prilog navedenom,

– Read address ulaz Memory jedinice mora biti share-ovan dodavanjem multipleksora sa 2 ulaza

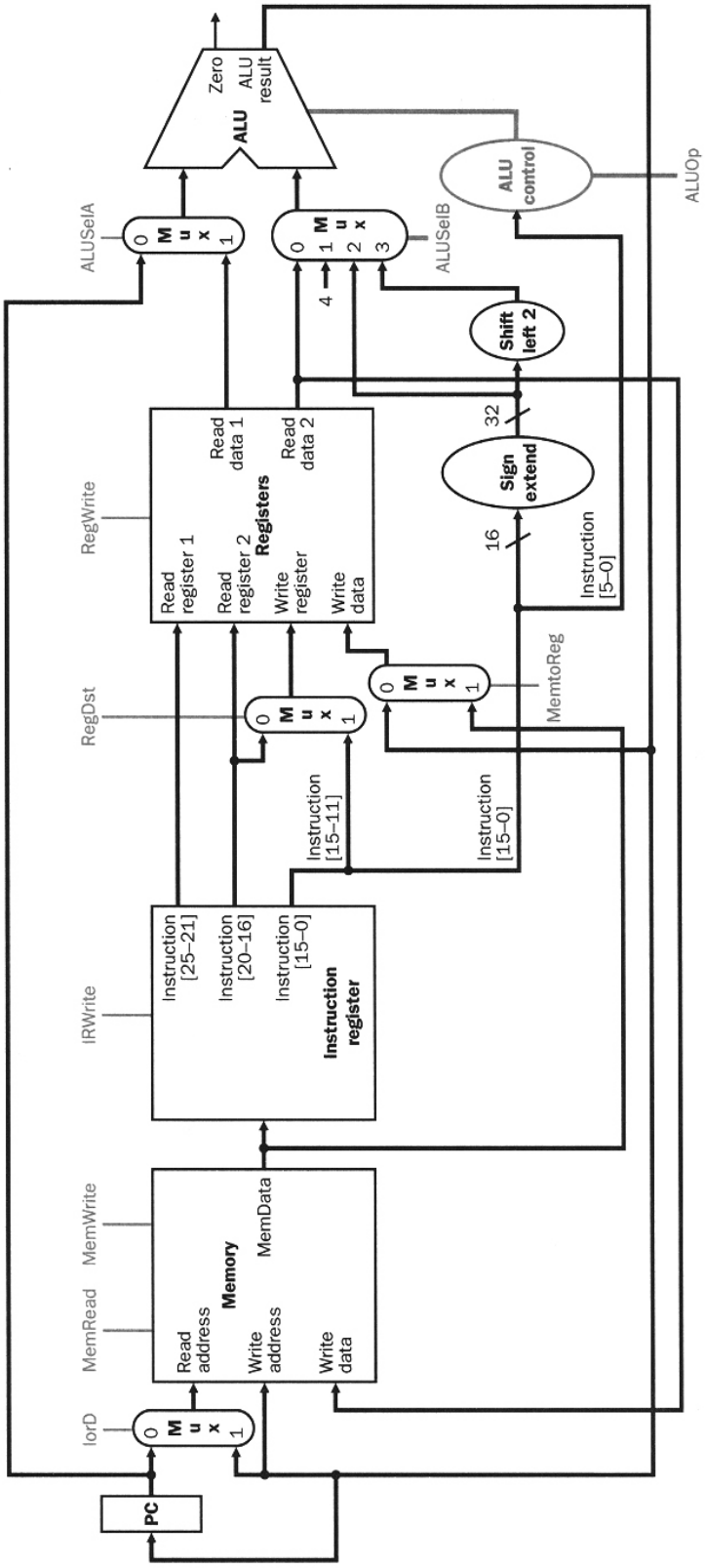
  - Na prvi ulaz ovog multipleksora dovodit će se sadržaj PC registra, u cilju obezbjedjivanja uslova za potencijalno uzimanje/čitanje instrukcije sa odgovarajuće lokacije,
  - Na drugi ulaz ovog multipleksora dovodit će se adresa izračunata od strane ALU (ALUOut), u cilju obezbjedjivanja uslova za potencijalno čitanje podatka sa odgovarajuće lokacije prilikom implementacije instrukcije *lw*.

  Multipleksor na Read address ulazu Memory jedinice pored 2 ulaza za odgovarajuće adrese treba da sadrži i 1 selekcioni ulaz, koji će biti nazvani shodno funkciji koju obavljaja, odnosno *IorD* (selektuje adresu lokacije u kojoj se nalazi instrukcija (I) ili podatak (D)).

– Na Write address ulaz Memory jedinice dovodit će se samo jedan signal – adresa izračunata od strane ALU (ALUOut), u cilju adresiranja lokacije u koju će potencijalno (prilikom implementacije instrukcije *sw*) biti upisan podatak doveden na Write data ulaz ove jedinice (na slici 9, zbog njene opštosti i ne ulaženja u detalje, Write data ulaz je označen sa Data).

Primijetimo da će multipleksor dodat na Read address ulazu Memory jedinice takodje biti novouveden u poredjenju sa jednotaktnom arhitekturom sa slike 8.

ZAPAŽANJE 3: Prilikom izvršavanje *lw* instrukcije, Memory jedinica upotrebljava se dva puta i to oba puta u funkciji čitanja (instrukcije, a potom i podatka). Naime, tom prilikom se, u toku prvog koraka, instrukcija čita iz Memory jedinice (u cilju njenog donošenja u proces izvršavanja), dok se tokom njenog izvršavanja (u IV koraku), iz Memory jedinice čita podatak. Primijetimo da prilikom čitanja podatka, instrukcija *lw* još uvijek nije izvršena, te da je potrebno izvršavanje još jednog koraka u cilju njenog kompletiranja (pogledaj tabelu 2 u sekciji 5.2). Drugim riječima, ukoliko se ne obezbijedi čuvanje instrukcije nakon njenog uzimanja/čitanja, pa do kraja njenog izvršavanja, podatak pročitan u IV koraku će prepisati izvršavanu instrukciju *lw* ili njene još uvijek neupotrijebljene djelove, odnosno ova instrukcija neće se moći izvršiti do kraja, te će dalji rad računara biti nepouzdan. Ova neželjena situacija prevazilazi se uvodjenjem dodatnog registra Instruction register (IR) koji je namijenjen čuvanju mašinskog koda instrukcije, od trenutka njenog uzimanja/čitanja iz Memory jedinice do njenog kompletiranja. Nakon toga, bezbjedno je, prilikom izvršavanja instrukcije *lw*, pročitati podatak iz Memory jedinice.



Slika 10. Detaljniji prikaz datapath-a za multitaktnu implementaciju sa naznačenim kontrolnim signalima memorijskih jedinica i prikazanih multipleksora. Primijetimo da na slici nije prikazan dio namijenjen upisivanju u PC registar, kao ni kontrolni signal upisivanja u PC registar.

Datapath za multitaktnu implementaciju koji sadrži značajno više detalja u odnosu na datapath sa slike 9, i koji je kreiran na osnovu prethodnih zapažanja 1–3, prezentiran je na slici 10.

ANALIZA: Pažljivom analizom datapath-a sa slike 10, pored prethodno detaljno razmatrane upotrebe samo jedne Memory jedinice (pogledaj zapažanje 2) i samo jedne ALU (pogledaj zapažanje 1), kao i uključivanja (u implementaciju) IR registra (pogledaj zapažanje 3), mogu se primijetiti i sljedeće činjenice koje dodatno razlikuju multitaktnu implementaciju sa slike 10 od jednotaktne implementacije sa slike 8:

1. Datapath sa slike 10 uključuje 2 dodatna multipleksora. Kao što je naznačeno u zapažanjima 1 i 2, to su multipleksori MUX 2/1 sa po jednim selekcionim ulazom. Jedan multipleksor MUX 2/1 dodat je na prvom ulazu ALU, a drugi na Read address ulazu Memory jedinice,
2. Postojećem multipleksoru MUX 2/1 na drugom ulazu ALU sa slike 8 dodata su 2 ulaza, tako da je na drugom ulazu ALU multipleksor MUX 4/1,
3. Ostali upotrijebljeni multipleksori na slici 10 (na Write register ulazu i Write data ulazu Registers jedinice) i kontrolni signali (*RegDst* i *MemtoReg*) dovedeni na njihove selekcione ulaze odgovaraju multipleksorima i pripadajućim kontrolnim signalima upotrijebljenim na istim ulazima Registers jedinice kod jednotaktne implementacije sa slike 8 (za funkcije ovih multipleksora pogledaj napomene 1 i 2 u sekciji 5.3.3 i razmatranja iz poglavlja 5.4),
4. Upotrijebljeni memorijski elementi (Memory jedinica, Registers jedinica i IR registar) posjeduju kontrolu upisivanja podataka. Nazivi njihovih kontrolnih signala (*MemWrite*, *RegWrite* i *IRWrite*) odgovaraju memorijskim elementima čije funkcionisanje kontrolišu i funkciji koju obavljaju,
5. Kao i u slučaju jednotaktne implementacije, pored kontrole upisivanja, Memory jedinica sadrži i kontrolni signal čitanja *MemRead* (potreba za uvodjenjem *MemRead* kontrolnog signala detaljno je razmatrana u napomeni 3 u poglavlju 5.4).

NAPOMENA 2: Na slici 10 nije uključen dio datapath-a koji obezbjedjuje upisivanje u PC registar, a shodno tome, ni kontrola upisivanja u PC registar. Ovaj dio datapath-a biće uključen kada i kontrolna jedinica i svi kontrolni signali koji utiču na kontrolu upisivanja u PC registar (treba predvidjeti kontrolu bezuslovnog i uslovnog upisivanja u ovaj registar u cilju implementacija instrukcije uslovnog skoka/grananja *beq*). Ipak, jednostavno je uočiti da na ulazu PC registra treba dodati multipleksor sa 3 ulaza (MUX 3/1), čiji ulazi treba da odgovaraju budućim potencijalnim sadržajima PC registra:

1. PC+4, kod implementacije instrukcija koje se sukcesivno izvršavaju (jedna-za-drugom),
2. Adresa Target, kod implementacije instrukcije uslovnog skoka/grananja *beq*,
3. Adresa bezuslovnog skoka, kod implementacije instrukcije *j*,

što je realizovano na slici 11. Primijetimo da ovaj multipleksor (MUX 3/1) treba da sadrži 2 selekciona ulaza na koja se dovodi 2-bitni *PCSource* kontrolni signal. Ipak, kontrolni signali i njihovo set-ovanje biće tema našeg interesovanja tokom kreiranja glavne kontrolne jedinice multitaktne implementacije.

5.5.2 Uključivanje registara za privremeno smještanje podataka u datapath za multitaktnu implementaciju

Kod jednotaktne implementacije (poglavlje 5.4) ne postoji mogućnost, ni potreba za uključivanjem regist(a)ra za privremeno smještanje podataka. Nema mogućnosti, jer jednotaktna implementacija raspolaže samo sa jednim taktnim intervalom tokom izvršavanja pojedinačne instrukcije i ne postoji dodatni taktni interval na čijoj bi se jednoj od ivica mogao u registru sačuvati podatak od prebrisavanja, da bi se isti kasnije (tokom izvršavanja iste instrukcije) mogao upotrijebiti. Nema potrebe, jer svaka akcija koja se obavlja mora biti izvršena u toku taktnog intervala kojim se raspolaže.

Sa druge strane, kod multitaktnie implementacije, svaka od instrukcija izvtršava se u toku trajanja nekoliko taktnih interval (3–5 taktnih intervala, zavisno od instrukcije koja se izvršava). Drugim riječima, kod višetaktne implementacija postoji (i te kako postoji) mogućnost za upotrebu regist(a)ra za privremeno smještanje podataka. Samo se postavlja pitanje da li postoji potreba za tim?

Odgovor na ovo pitanje već je dat u zapažanju 3 u sekciji 5.5.1. Naime IR registar, upotrijebljen prilikom dizajniranja datapath-a za multitaktnu implementaciju (slike 9 i 10), predstavlja registar za privremeno smještanje instrukcije (uzete/pročitane iz Memory jedinice u prvom taktnom intervalu i smještene u IR registar do kraja njenog izvršavanja – do III, IV ili V taktnog intervala, zavismo od trajanja instrukcije koja se izvršava). Potreba za uključivanjem IR registra u multitaktni implementaciju više je nego očigledna i detaljno je elaborirana u zapažanju 3 u sekciji 5.5.1.

Osim instrukcije, može postojati potreba za privremenim čuvanjem podat(a)ka ili rezultata ALU. Uočimo 2 kriterijuma neminovne upotreba registara za privremeno čuvanje podataka:

1. Instrukcija/podatak se uzima/čita iz memorijskog elementa u jednom taktnom intervalu, upotrebljava se u sljedećem ili u nekoliko sljedećih taktnih intervala, tokom njene/njegove upotrebe iz istog memorijskog elementa se uzima/čita novi podatak, a prvopročitana/i instrukcija/podatak nije u medjuvremenu sačuvana od prebrisavanja,
2. Rezultat se izračunava (od strane kombinacione logike – funkcionalnog elementa) u jednom taktnom intervalu, upotrebljava se u sljedećem ili nekoliko sljedećih taktnih intervala, tokom njegove upotrebe ista kombinaciona logika vrši nova izračunavanja, a prvodobijeni rezultat nije u medjuvremenu sačuvan od prebrisavanja.

Primijetimo da je uključivanje IR registra u multitaktnu implementaciju posljedica kriterijuma 1. od dva prethodno navedena kriterijuma neminovne upotrebe registra za privremeno čuvanje podataka. Razmotrimo što je sa rezultatima koje kreira ALU na svom izlazu i da li postoji potreba za uključivanjem dodatnih registara za privremeno smještanje njenih rezultata. ALU vrši izračunavanja PC+4, ciljne adrese grananja Target i operacije zahtijevane instrukcijom:

- PC+4 izračunava se u I taktnom intervalu izvršavanja instrukcija i na kraju istog taktnog intervala upisuje se nazad u PC registar (pogledaj tabelu 2). Drugim riječima, izračunati rezultat sačuva se u memorijskom elementu (PC registru) u toku istog taktnog intervala, tako da nema potrebe za njegovim čuvanjem i u nekom drugom memorijskom elementu.

  NAPOMENA 1: PC+4, nakon izračunavanja, upisuje se nazad u PC registar, jer će, u najvećem broju slučajeva, adresa instrukcije koja sljedeća treba da se izvrši biti upravo PC+4 (od ovoga pravila odstupaju samo instrukcije bezuslovnog skoka i instrukcije uslovnog skoka/grananja, ali ove potonje samo kada je uslov grananja zadovoljen).

- Ciljna adresa grananja Target izračunava se u II taktnom intervalu izvršavanja instrukcija, upotrebljava se u III taktnom intervalu ukoliko se izvršava *beq* instrukcija i ukoliko je zadovoljen uslov grananja (pogledaj tabelu 2) i nema smisla da se nakon izračunavanja upiše nazad u PC registar (time bi se izgibilo PC+4, izračunato u I taktnom intervalu i upisano u PC registar na kraju istog taktnog intervala). Medjutim, u III taktnom intervalu, ALU će biti upotrijebljena za izračunavanje operacije zahtijevane instrukcijom, pa čak i ukoliko je riječ o instrukciji *beq* (za poredjenje operanada Reg(rs) i Reg(rt)). Drugim riječima, ukoliko Target adresa, izračunata u II taktnom intervalu, ne bude sačuvana, biće prebrisana već u III taktnom intervalu i to tokom ispitivanja uslova grananja (prije konačne upotrebe adrese Target). Ovim je zadovoljen kriterijum 2. za uključivanje registra za privremeno smještanje ciljne adrese grananja. Registar je uključen u implementaciju, kao što je prikazano na slici 11, i nazvan je Target, shodno podatku (adresi) koji čuva od prebrisavanja (pogledaj sliku 11).

  NAPOMENA 2: Izračunata ciljna adresa grananja ne upisuje se nazad u PC registar nakon njenog izračunavanja (na kraju II taktnog intervala). Naime, time bi se prebrisala vrijednost PC+4 upisana na kraju I taktnog intervala, a sa druge strane, ne postoji garancija da će izračunata ciljna adresa grananja uopšte biti upotrijebljena do kraja izvršavanja instrukcije (upotrebljava se samo ukoliko se izvršava *beq* instrukcija, a to još uvijek ne znamo za vrijeme izračunavanja ciljne adrese grananja, i samo ukoliko je uslov grananja zadovoljen).

- Operacija zahtijevana instrukcijom izvršava se u III taktnom intervalu (pogledaj tabelu 2), a dobijeni rezultat upotrebljava se u IV taktnom intervalu (prilikom implementacije instrukcija R-tipa i *sw* instrukcije – pogledaj tabelu 2) ili u IV i u V taktnom intervalu (prilikom

implementacije *lw* instrukcije – pogledaj tabelu 2). Medjutim, na ulaze ALU nalaze se isti potencijalni operandi tokom svakog od navedenih taktnih intervala. Naime, instrukcija, odnosno sva njena polja (pa i ona polja čijim sadržajima se označavaju registri iz Registers jedinice koji sliže kao potencijalni operandi ALU i address/immediate/offset polje koje takodje može obavljati ulogu operanda ALU) sačuvani su u IR registru i ne mogu biti izmijenjeni do kraja izvršavanja instrukcije (na koncu, to je namjena IR registra). Sa druge strane, selekciju operanada ALU zahtijevanih instrukcijom obavljaju kontrolni signali *ALUSelA* i *ALUSelB* multipleksora na ulazima ALU, dok selekciju operacije ALU zahtijevane instrukcijom obavljaju *ALUOp* kontrolni signali, a vrijednosti kontrolnih signala postavlja/set-uje glavna kontrolna jedinica koju ćemo mi dizajnirati. Dakle, selekcija operanada i funkcionisanje ALU je pod potpunom kontrolom dizajnera glavne kontrolne jedinice, tako da se do kraja izvršavanja instrukcije može obezbijediti zadržavanje istog rezultata na izlazu ALU, odnosno neprebrisavanje rezultata ALU, dobijenog u III taktnom intervalu. Drugim riječima, ne postoji potreba za uključivanjem dodatnog registra za privremeno smještanje rezultata ALU.

SUMARUM: Na slici 11, prikazan je kompletni dizajn za multitaktnu implementaciju, koji uključuje 2 registra za privremeno smještanje podataka/rezultata ALU: Instruction registar (IR – za detalje pogledati zapažanje 3 u sekciji 5.5.1) i Target registar za privremeno smještanje ciljne adrese grananja.

NAPOMENA 3: Arhitektura za multitaktnu implementaciju, prikazana na slici 11, uključuje glavnu kontrolnu jedinicu sa njenim ulazima, op poljem instrukcije (Op [5–0]=Instruction [31–26]), i izlazima–kontrolnim signalima–povezanim sa selekcionim ulazima upotrijebljenih multipleksora i kontrolnim ulazima upotrijebljenih memorijskih elemanata. Dodata je i kontrola upisivanja u PC registar (pogledaj napomenu 2 u sekciji 5.5.1). Ipak, dizajniranje glavne kontrolne jedinice i kontrola funkcionisanja datapath-a biće detaljno razmatrane u sljedećoj sekciji.

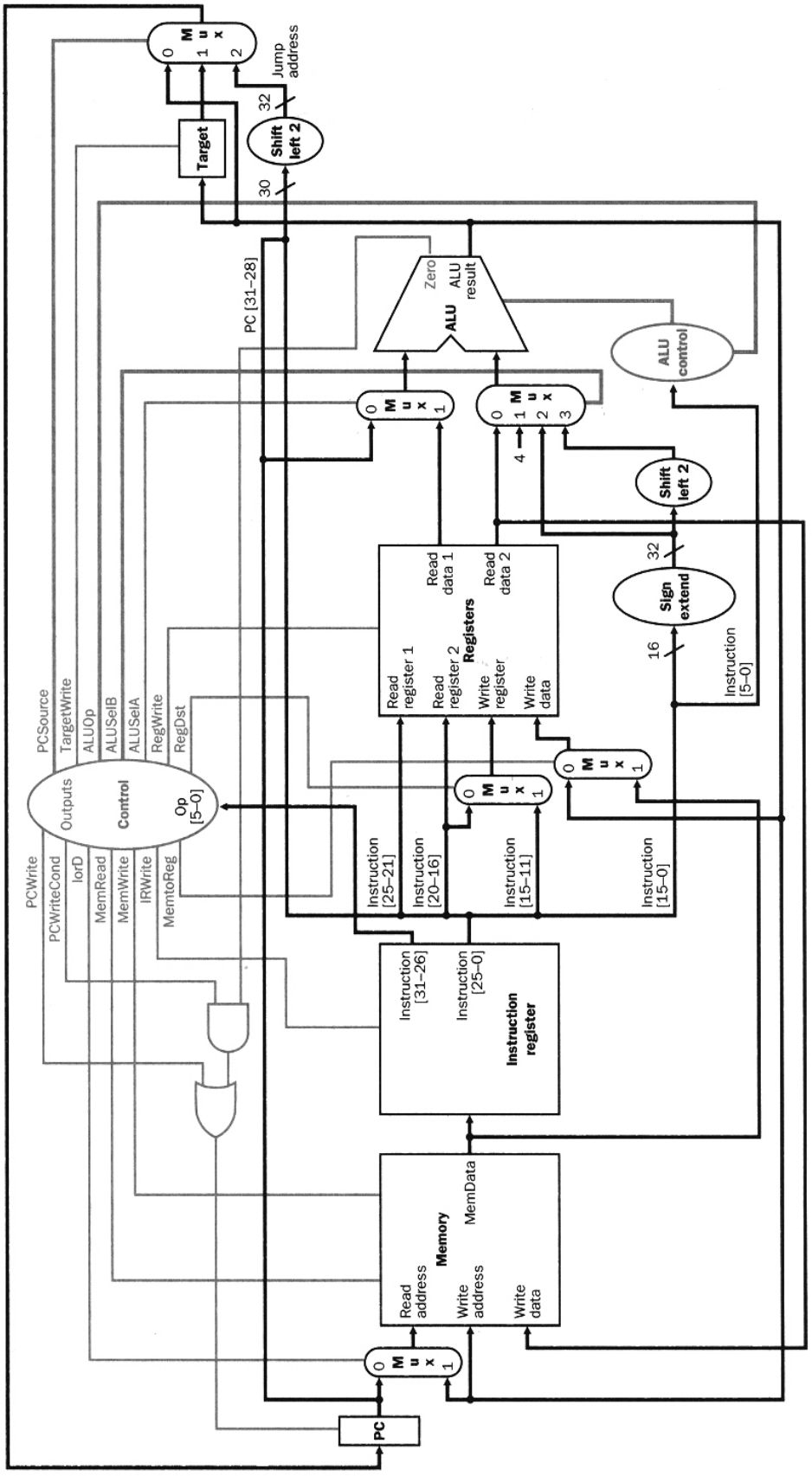
NAPOMENA 4: Na slici 11, prikazana je i realizacije instrukcije bezuslovnog skoka *j*. Realizuje se formiranjem 32-bitne adrese bezuslovnog skoka (Jump address na slici 11) i njenim dovodjenjem na ulaz br. 2 multipleksora koji se nalazi na ulazu PC registra (pogledaj napomenu 2 iz sekcije 5.5.1). Jump address se formira kombinacijom najviša 4 bita tekućeg sadržaja PC registra, PC [31–28], i 26-bitnog adresnog polja instrukcija J-tipa, Instruction [26–0], te shift-ovanjem tako dobijene 30-bitne adrese za 2 mjesta u lijevu stranu, u skladu sa razmatranjima iz sekcije 5.3.5, napomenom iz sekcije 5.3.4, te napomenom 7 i konvencijom iz poglavlja 3.7.

Detaljno poredjenje (s aspekta hardware-ske složenosti) jednotaktne implementacije, date na slici 8, i multitaktne implementacije, prikazane na slici 11, formirano na osnovu analize iz sekcije 5.5.1 i zapažanja iz napomene 2 iz iste sekcije, sumirano je u tabeli 13.

Tabela 13. Jednotaktna (SCI) vs multitaktna implementacija (MCI) s aspekta hardwere-ske složenosti.

Uredjaji	SCI	MCI
Funkcionalni jedinice	ALU + 2×Add	ALU $\begin{pmatrix} +\text{MUX 2/1 na I ulazu ALU} \\ +2 \text{ ulaza na MUX na II ulazu ALU} \end{pmatrix}$
Memorijske jedinice	Instruction + Data	Memory (+ MUX 2/1 na Read asddress ulazu)
Privremeni registri	0	2 (IR + Target)

ZAKLJUČNA RAZMATRANJA: Multitakna implementacija sa slike 11 uključuje po jednu veliku funcionalnu (ALU), odnosno memorijsku (Memory) jedinicu, za razliku od jednotaktne implementacije koja uključuje ALU i 2 dodatna sabirača, te 2 velike memorijske jedinice (Instruction i Data memory). Ovo se "plaća" (od strane multitaktne implementacije) sa 2 dodata multipleksora MUX 2/1, 2 dodata ulaza na još jednom multipleksoru MUX 2/1, te sa 2 registra za privremeno smještanje instrukcije/rezultata ALU (IR i Target registar). Medjutim, multipleksori i registri su mali i veoma jeftini elementi, koji, uz to, zahtijevaju malu količinu energije u poredjenju sa ALU i velikim memorijskim jedinicama. Stoga, možemo zaključiti da multitaktna implementacija optimizira ne samo hardware-sku složenost i gabarite sistema, već i njegovu cijenu i potrošnju energije, kao i vrijeme izvršavanja grupe instrukcija i izvršavanih programa (kao što je pokazano u sekciji 5.5). Shodno tome, multitaktna implementacija predstavlja praktično aplikabilno rješenje za dizajniranje procesora.



Slika 11. Kompletni dizajn za multitaktnu implementaciju.