5 PROCESOR

Centralna procesorska jedinica (eng. *Central Processing Unit* – CPU) ili skraćeno procesor je najznačajnija cjelina računarskih arhitektura/sistema. Uopšteno, procesor se sastoji iz Datapath-a (ALU, 32 procesorska registra, dodatnog kontrolnog interfejsa i medjusobnih veza ovih elemenata) i kontrolne jedinice. Kao što je elaborirano u poglavlju 4, ALU obavlja operacije koje se zahtijevaju kodom izvršavanih instrukcija. Operacije se izvršavaju nad operandima koji se nalaze u procesorskim registrima ($0–$31), a elementi kontrolnog interfejsa (multipleksori, dodatna logička kola, dekoderi) obezbjedjuju pravilan odabir i protok operanada, kao i podataka uzetih/pročitanih iz memorije računara i podataka koje je potrebno upisati u memoriju računara. Kontrolna jedinica postavlja kontrolne signale svih elemenata kontrolnog interfejsa, operativne memorije računara i ostalih upotrijebljenih memorijskih elemenata. Na ovaj način, kontrolna jedinica suštinski upravlja ispravnim funkcionisanjem računara kao sistema/cjeline.

Kada se govori o performansama računara obično se misli na:

- brzinu rada procesora, odnosno frekvenciju njegovog taktnog signala.

Medjutim, frekvencija rada procesora je samo jedna od tri faktora koja dominnantno utiču na performansama računara. Preostala dva su:

- broj instrukcija koje je potrebno izvršiti u assembler-skoj formi programa za izvršavanje odredjene akcije,
- broj taktnih intervala koje je neophodno izvršiti po instrukciji.

Broj instrukcija koje se izvršavaju u assembler-skoj formi odredjen je compiler-om, odnosno skupom instrukcija kojima compiler raspolaže. Naime, bez obzira da li se radi o programu koji se izvršava u nekom od viših programskih jezika, ili je riječ o akciji koja se izvršava pod komandom operativnog sistema, ovaj program/akcija mora biti zapisana/zadata u višem programskom jeziku. Potom se program/akcija prevode (kompajliraju) u assembler-sku formu. Assembler-ska forma će zauzeti manje linija koda (podsjetimo se, u assembler-skoj formi, u svakoj liniji koda može biti zapisana tačno jedna instrukcija) ukoliko compiler raspolaže širim skupom i moćnijim instrukcijama. Uz to, kada se kaže skup instrukcija u assembler-skoj formi misli se na arhitekturu kreiranu za podrazumijevani skup instrukcija. Podsjetimo, u poglavlju 3, razmatrali smo instrukcije u assemblerskoj formi. Od arhitekture neophodne za implementaciju intrukcija uvedenih u assembler-skoj formi u poglavlju 3, do sada je dizajnirana ALU (poglavlje 4).

Frekvencija rada procesora i broj taktnih intervala po instrukciji odredjeni su implementacijom procesora. Oni si tema izučavanja u ovom poglavlju. U tom cilju, biće razmatrane jednotaktna (ili prosta) i višetaktna (odnosno multitaktna) implementacija procesora za odredjeni skup instrukcija.

Implementirane će biti instrukcije koje predstavljaju srž MIPS seta instrukcija, i to:

- Data-transfer, odnosno memory-reference instrukcije *lw* i *sw*,
- Aritmetičko-logičke instrukcije (instrukcije R-ipa) *add*, *sub*, *and*, *or* i *slt*,
- Instrukciju uslovnog skoka/grananja *beq*, i
- Instrukciju bezuslovnog skoka *j*.

NAPOMENA: Ovim setom, odnosno podskupom instrukcija nijesu obuhvaćene sve MIPS instrukcije, čak ni sve instrukcije razmatrane u poglavlju 3. Medjutim, razmatrani podskup je reprezentativan i predstavlja osnovu za ilustraciju ključnih principa koji se upotrebljavaju u kreiranju datapath-a i kontrolne jedinice.

POSLJEDICA: Instrukcije, koje nijesu obuhvaćene prethodnim setom instrukcija, mogu se implementirati na način veoma sličan onom koji će biti prezentiran prilikom dizajniranja instrukcija iz prethodno navedenog seta. To će na kraju poglavlja biti ilustrovano na primjerima immediate instrukcija, kada će se postojeći procesor (za prethodno navedeni set instrukcija) dopuniti i korigovati tako da omogući implementaciju immediate instrukcija.

VAŽNO: Najveći broj koncepata koji će biti upotrebljeni u dizajniranju procesora za nevadeni set instrukcija predstavljaju koncepte koji se upotrebljavaju u dizajniranju širokog spektra računara, od računara sa viskim performansama, do mikroprocesora opšte i specijalizovane namjene.

5.1 PREGLED MIPS INSTRUKCIJA U ASSEMBLER-SKOM I MAŠINSKOM KODU

Prije nego započnemo dizajniranje, napravimo kratak pregled MIPS instrukcija i formata njihovog zapisivanja u assembler-skoj i mašinskoj formi, tabela 1.

Tabela 1. Posmatrane MIPS instrikcije i formati njihovog zapisivanja.

Aritmetičko-logičke instrukcije – *add*, *sub*, *mult*, *slt*, *sll*, *srl*…

Sintaksa: *add* $x, $y, $z
Format zapisivanja: R-tip
Šematski prikaz:

Polje:	op	rs	rt	rd	shamt	funct
Br. bitova:	6	5	5	5	5	6
Sadržaj:	0	y	z	x	n<31	32,34, …

Data-transfer instrukcije – *lw*, *sw*

Sintaksa: *lw* (*sw*) $x, Astart($y)
Format zapisivanja: I-tip
Šematski prikaz:

Polje:	op	rs	rt	Address
Br. bitova:	6	5	5	16
Sadržaj:	35, 43	y	x	Astart

Immediate instrukcije *addi*, *muli*, *slti* ...

Sintaksa: *addi* $x, $y, C
Format zapisivanja: I-tip
Šematski prikaz:

Polje:	op	rs	rt	Immediate
Br. bitova:	6	5	5	16
Sadržaj:	8	y	x	C

Branch instrukcije (instrukcije uslovnog skoka/grananja) – *beq*, *bne*

Sintaksa: *beq* (*bne*) $x, $y, offset
Format zapisivanja: I-tip
Šematski prikaz:

Polje:	op	rs	rt	Immediate
Br. bitova:	6	5	5	16
Sadržaj:	4, 5	x	y	Offset

Instrukcije bezuslovnog skoka – *j*, *jal*

Sintaksa: *j* address
Format zapisivanja: J-tip
Šematski prikaz:

Polje:	op	Address
Br. bitova:	6	26
Sadržaj:	2, 3	Address

ZABILJEŠKA: Polja mašinskog koda zapisivanja instrukcija zauzimaju tačno odredjena mjesta u kodu, tabela 1, i mogu se predstaviti shodno bitovima koja zauzimaju. Na primjer, op polju odgovara najviših 6 bitova instrukcije, ili op=Instruction [31–26], kao i rs=Instruction [25–21], rt=Instruction [20–16], rd=Instruction [15–11], shamt=Instruction [10–6], address/immediate/offset=Instruction [15–0], funct=Instruction [5–0], i address polje instrukcija bezuslovnog skoka address=Instruction [25–0]. Na šematskim prikazima, polja instrukcija će, po pravilu, biti označavana na ovaj način.

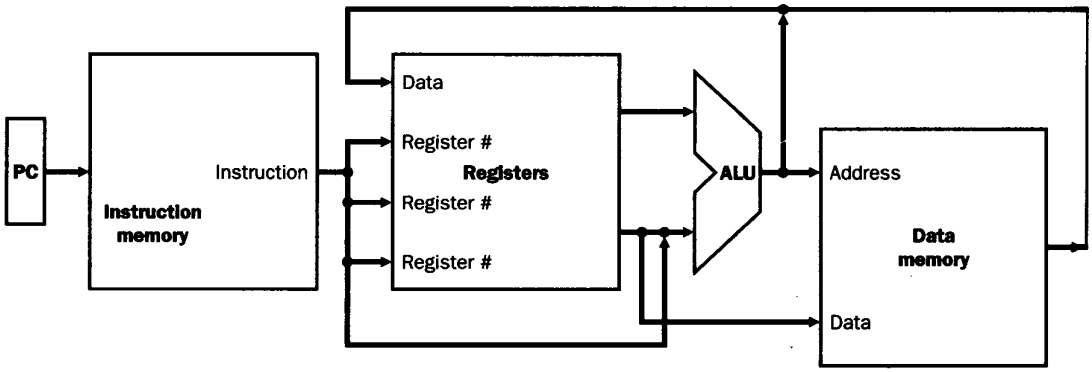
5.2 OPŠTI POGLED NA IMPLEMENTACIJU

Na osnovu znanja stečenih u oblasti programiranja u assembler-u, te prezentiranog kratkog pregleda MIPS inskrukcija i formata njihovog zapisivanja u assembler-skom i u mašinskom obliku, tabela 1, mogu se zapaziti koraci koje svaka pojedninačna instrukcija mora obaviti tokom izvršavanja, kao što je prikazano u tabeli 2. U tabeli 2 dati su detalji izvršavanja MIPS instrukcija po koracima. Medjutim, za sada ćemo pažnju posvetiti samo koracima i osnovnim razlozima za njihovo obavljenje, bez dubljeg ulaska u detalje. Detalji izvršavanja po koracima detaljno će biti razmatrani kasnije.

Tabela 2. Neophodni koraci koje je po trebno izvršiti u cilju implementacije MIPS instrukcija.

<table>
<tr><th>Korak</th><th>Data-transfer instrukcije (lw, sw) – I tip</th><th>Aritmetičko-Log. instrukcije (R-tip)</th><th>Branch instrukcije (beq) – I tip</th><th>Bezuslovni skok (j) – J tip</th></tr>
<tr><td>I</td><td colspan="4">Akcija: Uzimanje instrukcije iz memorije i njeno donošenje u proces izvršavanja (IR←Mem[PC])
Paralelna akcija: Inkrementiranje sadržaja PC-a (PC←PC+4)</td></tr>
<tr><td>II</td><td colspan="4">Akcija: Dekodiranje instrukcije – čitanje sadržaja registara označenim poljima rs i rt, kao i sadržaja polja op, shamt, funct, address, offset, … (nepotreno postavljati kontrolne signale, pošto se čitanje memorijskih elemenata, sa izuzetkom memorije računara, ne kontroliše, već se kontroliše samo njihovo upisivanje)
Paralelna akcija: Izračunavanje ciljne adrese grananja (Target←(PC+4)+sign_extend(offset)<<2)</td></tr>
<tr><td>III</td><td colspan="4">Upotreba ALU za izračunavanje operacije zadate instrukcijom i generisanje izlaza ALU (ALUOut, Zero, Overflow)</td></tr>
<tr><td></td><td>ALUOut←
←Reg(rs)+sign_extend(address)</td><td>ALUOut←Reg(rs) <u>op</u> Reg(rt)
(<u>op</u> – operacija zahtijevana instrukcijom)</td><td>PC←Target samo ako je Reg(rs)=Reg(rt) (Zero=1)</td><td>PC←(PC[28–31]+address)<<2</td></tr>
<tr><td>IV</td><td colspan="4">Upotreba rezultata ALU u cilju kompletiranja naredbe</td></tr>
<tr><td></td><td>lw
Čitanje Mem[ALUOut]</td><td>sw
Mem[ALUOut] ←Reg(rt)</td><td colspan="2" rowspan="2">Reg(rd)←ALUOut</td></tr>
<tr><td>V</td><td>Reg(rt)← Mem[ALUOut]</td><td colspan="3"></td></tr>
</table>

Da bi instrukcija mogla biti izvršena, ona najprije mora biti pročitana (iz memorije računara) i dekodirana. Čitanjem se instrukcija dovodi u proces izvršavanja, dok se njenim dekodiranjem čitaju polja instrukcije zapisane u mašinskom kodu. Očitavanje polja neophodno je za izvršavanje svake pojedinačne instrukcije. Na primjer, tek nakon očitavanja vrijednosti zapisane u polju op, dolazi se do saznanja o kojoj instrukciji je riječ i što je sve neophodno preduzeti u cilju njenog daljeg izvršavanja. Slično je i sa ostalim poljima instrukcije: očitavanjem obilježja registara utvrdjuju se operandi nad kojima se obavlja operacija zahtijevana instrukcijom, očitavanjem adresnog/immediate/offset polja utvrdjuje se početna adresa niza (kod implementacije instrukcija *lw* i *sw*), konstanta koja služi kao operand (kod immediate instrukcija), odnosno PC relativne adrese (kod instrukcija *beq* i *bne*), ... Stoga su prva dva koraka izvršavanja (čitanje instrukcije i njeno dekodiranje) obavezna. Istovremeno,



Slika 1. Sažeti pogled na MIPS implementaciju različitih tipova instrukcija.

oni su zajednički koraci koje mora obaviti svaka instrukcija na početku svog izvršavanja.

Paralelno sa čitanjem instrukcije i njenim dekodiranjem, u zajeničkim koracima (prvom i drugom) izvršavaju se i akcije koje mogu biti od koristi za kompletiranje pojedinih instrukcija (i skratiti njihovo izvršavanje za jedan korak), a čije obavljanje ne nanosi štetu ni izvršavanju drugih instrukcija (instrukcija kojima ove akcije nijesu namijenjene). Akcije koje se izvršavaju u paraleli sa čitanjem i dekodiranjem instrukcije odnose se na kreiranje adrese instrukcije koja sljedeća treba da se izvrši. U prvom koraku, paralelno čitanju instrukcije, inkrementira se sadržaj PC registra (uvećava se za 4 byte-a, PC←PC+4, da bi se sadržaj PC registra postavio na adresu instrukcije koja je u memoriji računara zapisana odmah nakon tekuće instrukcije – o ovoj akciji je već bilo riječi prilikom razmatranja PC-relativnog adresiranja). U drugom koraku, paralelno dekodiranju pročitane instrukcije, pronalazi se ciljna adresa grananja/skoka (tzv. Target adresa), koja može biti od koristi ukoliko se zaključi, nakon dekodiranja, da je riječ o instrukcijama uslovnog skoka/grananja, te ukoliko je uslov grananja zadovoljen.

Nakon dekodiranja instrukcije, ima se uvid o kojoj instrukciji je riječ i može se znati što je potrebno napraviti u cilju kompletiranja svake pojedinačne instrukcije. Uopšteno rečeno, treći korak izvršavanja odnosi se na upotrebu ALU za izračunavanje operacije zadate instrukcijom i generisanje izlaza ALU (ALUOut, Zero, Overflow), dok je četvrti (i eventualni peti) korak namijenjen upotrebi rezultata ALU dobijenih u trećem koraku.

Instrukcije *lw* i *sw* upotrebljavaju ALU u cilju izračunavanja adrese memorijske lokacije koja treba da učestvuje u transferu podataka sa registarom, čije obilježje je upisano u rt polju instrukcije. Nakon toga, u četvrtom (i eventualnom petom) koraku vrši se transfer podataka iz memorije računara u registar označen poljem rt (kod instrukcije *lw*), odnosno transfer podataka u inverznom/obrnutom smjeru (kod instrukcije *sw*).

Aritmetičko-logičke instrukcije (instrukcije R-tipa) upotrebljavaju ALU u cilju obavljanja operacije zahtijevane izvršavanom instrukcijom (dizajnirana je, u poglavlju 4, ALU za obavljanje operacija AND, OR, sabiranje, oduzimanje, Set-on-less-than). Nakon toga, u četvrtom koraku, rezultat zahtijevane operacije, izračunat u trećem koraku, upisuje se u registar označem rd poljem instrukcije.

Instrukcije uslovnog skoka/grananja (branch instrukcije) upotrebljavaju ALU za 2 namjene:

1. Za izračunavanje ciljne adrese grananja (tzv. Target adrese),
2. Za ispitivanje ispunjenosti uslova jednakosti operanada, kada je potrebno da je Zero=1 (prilikom implementiranja instrukcije *beq*), odnosno za ispitivanje ispunjenosti uslova nejednakosti operanada, kada je potrebno da je Zero=0 (prilikom implementiranja instrukcije *bne*).

Primijetimo da se izračunavanje ciljne adrese grananja (Target adrese) vrši u drugom koraku, paralelno sa dekodiranjem instrukcije, dok se ispitivanje/provjera ispunjenosti uslova grananja i smještanje (u PC registar) adrese instrukcije koja treba da se izvrši nakon instrukcije *beq*/*bne* izvršava u trećem koraku.

Instrukcija bezuslovnog skoka *j* ne upotrebljava ALU. Za njeno izvršavanje, potrebno je kreirati (u PC registru) adresu na kuju treba izvršiti skok, o čemu će više riječi biti kasnije (nakon dizajniranja datapath-a za ostale razmatrane instrukcije).

Sažeti pogled na MIPS implementaciju različitih tipova instrukcija prikazan je na slici 1. Ona pokušava da obuhvati sve prethodna razmatranja. U PC registru nalazi se adresa instrukcije koja treba da se izvrši. PC registar svojim sadržajem adresira lokaciju Instruction memory sa koje će instrukcija biti pročitana i donesena u proces izvršavanja. Nakon toga, instrukcija se dekodira. Sadržajima polja rs i rt instrukcije označavaju se operandi u registrarskoj jedinici/fajlu (na slici 1 nazvanoj Registers, koja obuhvata procesorske registre $0–$31). U polju address/immediate/offset nalazi se početna adresa (kod instrukcija *lw* i *sw*), konstanta (kod immediate instrukcija), odnosno PC-relativna adresa (kod instrukcija *beq* i *bne*). Nakon dekodiranja, svaka instrukcija ima svoj tok izvršavanja:

- Operandi pročitani/uzeti sa izlaza Registers jedinice, iz registara označenih poljima rs i rt instrukcija R-tipa (aritmetičko-logičke instrukcije), dolaze na ulaz ALU, koja nad njima obavlja operaciju zahtijevanu izvršavanom instrukcijom. Rezultat dobijen na izlazu ALU upisuje se nazad u Registers jedinicu, u registar označen poljem rd instrukcije,
- Operand pročitan/uzet sa izlaza Registers jedinice, iz registra označenog poljem rs instrukcije, i adresa iz polja address instrukcija *lw* i *sw* predstavljaju operande nad kojima ALU izračunava adresu lokacije u Data memory koja će vršiti razmjenu podataka sa registrom označenim poljem rt instrukcije:
  - Ukoliko je riječ o instrukciji *lw*, adresa izračunata od strane ALU odgovara adresi lokacije sa koje će se pročitati podatak (Read address). Podatak pročitan iz Data memory upisuje se nazad u Registers jedinicu, u registar označen poljem rt instrukcije,
  - Ukoliko je riječ o instrukciji *sw*, adresa izračunata od strane ALU služi kao adresa za upisivanje podataka (Write address) u lokaciju kojoj izračunata adresa odgovara. Podatak, koji se upisuje u ovu lokaciju, uzima se sa izlaza Registers jedinice, iz registra označenog poljem rt instrukcije,
- Operandi pročitani/uzeti sa izlaza Registers jedinice, iz registara označenih poljima rs i rt instrukcija uslovnog skoka/grananja *beq*/*bne*, dovode se na ulaze ALU, koja provjerava jednakost/nejednakost sadržaja ovih registara. Zero izlaz ALU predstavlja rezultat koji će upravljati postavljanjem budućeg sadržaja PC registra. Ovaj dio nije prikazan na slici 1, jer je suviše detaljan za sažeti pogled prikazan na ovoj slici.

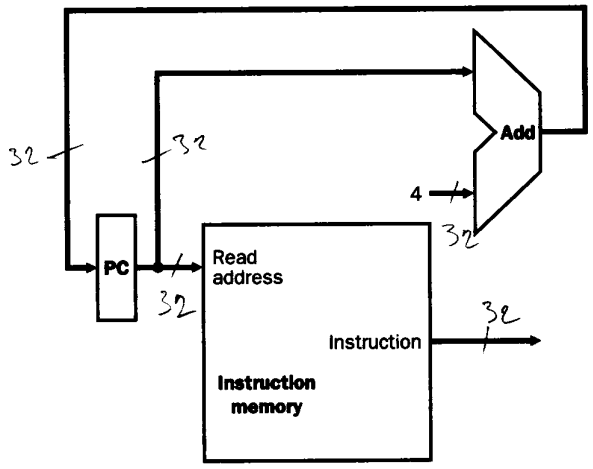
Primijetimo, odmah na strartu, da su drugi ulaz ALU i ulaz Data Registers jedinice podijeljeni (eng. *share*-ovani) za upotrebu izmedju različitih instrukcija:

- Drugi ulaz ALU može biti sadržaj registra/operand označen poljem rt instrukcija R-tipa, ali može biti i sadržaj address polja instrukcija *lw*/*sw*.
- Ulaz Data Registers jedinice može biti rezultat ALU (kod instrukcija R-tipa), ali može biti i podatak pročitan iz memorije računara (kod instrukcije *lw*).

Neophodnost podjele prethodno navedenih ulaza ALU i Registers jedinice može se primijetiti na slici 1, iako ovo nije i formalno prezentirano na slici. Share-ovanje/dijeljenje se obavlja postavljanjem multipleksora na odgovarajućim ulazima funkcionalnih jedinica, i to multipleksora čija veličina (broj ulaza) odgovara namjenama izmedju koji se funkcionalne jedinice dijele/share-uju.

5.3 SASTAVNI DJELOVI DATAPATH-A

Na osnovu navoda o potrebi share-ovanja/podjele pojedninih funkcionalnih jedinica, te njenog neprikazivanja na slici 1 može se primijetiti da je prikaz dat na slici 1 suviše sažet i da prikazuje nedovoljno detalja. U prethodnom tekstu pokušano je da se opiše svaki detalj prikazan na ovoj slici. Medjutim, za više detalja moramo proći kroz djelove datapath-a namijenjene izvršavanju pojedinačnih instrukcija, kroz njihovu integraciju u jedinstveni datapath, dodavanje kontrolnog interfejsa na svim



Slika 2. Dio datapath-a namijenjen donošenju instrukcije i inkrementiranju sadržaja PC registra.

funkcionalnim jedinicama koje je potrebno share-ovati/dijeliti i, na koncu, kroz dizajniranje kontrolne jedinice, što će biti razmatrano do detalja u nastavku.

5.3.1 Dio datapath-a namijenjen čitanju instrukcije i inkrementiranju sadržaja PC registra

Prvim (zajedničkim) korakom za implementaciju instrukcija predvidjeno je čitanje instrukcije iz memorije računara (odnosno, njeno donošenje u proces izvršavanja) i, u paraleli, inkrementiranje sadržaja PC registra. Dio datapath-a namijenjem izvršavanju ovih akcija prikazan je na slici 2.

Podsjetiimo, PC registar sadrži adresu instrukcije koja treba da se izvrši. Shodno tome, PC registar svojim sadržajem adresira, na "Read address" ulazu Instruction memory (memorijska jedinica namijenjena čuvanju instrukcija), lokaciju u kojoj je instrukcija zapisana. Na izlazu Instruction memory dobija se pročitana instrukcija koju treba izvršiti u narednim koracima. U paraleli, sadržaj PC registra dovodi se na prvi ulaz 32-bitnog sabirača, na čiji drugi ulaz se dovodi konstanta 4 da bi sabirač obavio operaciju PC+4. Rezultat ove operacije upisuje se, sa prvom sljedećom aktivnom ivicom takta, nazad u PC registar. Kontrola upisivanja podataka u memorijske elemente (time i u PC registar) nije predmet našeg razmatranja u ovom trenutku, te stoga nije ni prikazana na slici 2. Kontrola upisivanja će biti razmatrana kasnije prilikom dizajniranja kontrolne jedinice.

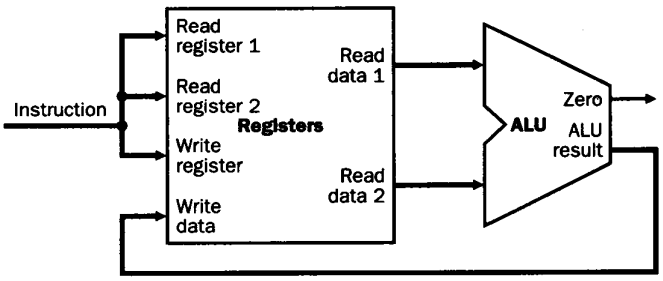
Primijetimo da je sadržaj PC registra 32-bitni, kao i kod pročitane instrukcije, u skladu sa razmatranjima iz poglavlja 3 (razmatramo, na koncu, 32-bitnu MIPS arhitekturu). Shodno 32-bitnom dizajnu sabirača, konstanta 4 koja se dovodi na njegov drugi ulaz takodje mora biti 32-bitna ($00...0100_{(2)}$). Primijetimo da je šemetski simbol sabirača veoma sličan simbolu ALU, uz navedenu oznaku "Add" na sabiraču. Na izlazu Instruction memory nalazi se instrukcije pročitana i donesena u proces izvršavanja. Ona (njena polja) će predstavljati ulaz ostalih djelova datapath-a.

5.3.2 Dio datapath-a namijenjen kreiranju aritmetičko-log. instrukcija (instrukcija R-tipa)

Podsjetimo na format zapisivanja aritmetičko-logičkih instrukcija (instrukcija R-tipa), tabela 1. Poljima rs i rt ovih instrukcija označeni su procesorski registri čiji sadržaji predstavljaju operande nad kojima ALU izvršava operaciju zahtijevanu instrukcijom (ALU je dizajnirana za obavljenje operacija AND, OR, sabiranje, oduzimanje ili Set-on-less-than). Rezultat zahtijevane operacije, sa izlaza ALU, upisuje se nazad u procesorski registar označen poljem rd instrukcije.

Procesorski registri ($0–$31) čiji sadržaji predstavljaju operande ALU nalaze su u Registers jedinici, kao i registar u koji se upisuje rezultat zahtijevane operacije. Shodno tome, Registers jedinica treba da posjeduje ulaze za adresiranje registara čiji sadržaji će predstavljati operande, kao i ulaz za adresiranje registra u koji treba da se upiše rezultat operacije:

– Ulazi za adresiranje registara čiji sadržaji predstavljaju operande nazivaju se "Read register 1" i "Read register 2" adresnim ulazima (uzimanjem operanada iz Registers jedinice vrši se



Slika 3. Dio datapath-a namijenjen kreiranju instrukcija R-tipa.

čitanje (eng. *Read*) odgovarajućih registara). Shodno navodima na početku ove sekcije, na ulaze Read register 1 i Read register 2 dovode se redom sadržaji polja rs i rt instrukcije R-tipa. Ovim adresnim ulazima odgovaraju izlazi "Read data 1" i "Read data 2" Registers jedinice, sa kojih se uzimaju (čitaju – eng. read) sadržaji registara označenih na "Read register 1" i "Read register 2" adresnim ulazima. Drigim riječima, svakom od operanada (podataka) koje treba pročitati iz Registers jedinice odgovara po jedan ulaz i izlaz – jedan adresni ulaz i njemu odgovarajući izlaz za pročitani podatak.

– Ulaz za adresiranje registra u koji treba da se **upiše** (eng. *Write*) rezultat operacije naziva se "Write register" adresnim ulazom. Shodno navodima na početku ove sekcije, na Write register adresni ulaz dovodi se sadržaj polja rd instrukcija R-tipa. Ovom adresnom ulazu odgovara ulaz "Write data" koji se upotrebljava za donošenje rezultata (sa izlaza ALU) u cilju njegovog upisivanja (eng. write) u registar označen na "Write register" adresnom ulazu. Drugim riječima, podatku (rezultatu sa izlaza ALU) koji treba upisati u Registers jedinicu odgovaraju 2 ulaza – adresni i njemu odgovarajući ulaz za podatak koji je potrebo upisati.

Primijetimo da su adresni ulazi Regisetrs jedinice (Read register 1, Read register 2 i Wrire register) 5-bitni, pošto služe za adresiranje 32 različita registra ($0–$31), a 32 različita obilježja ovih registara mogu biti zapisana sa 5 bitova (2^5=32). Sa druge strane, izlazi za čitanje podataka (Read data 1 i Read data 2), kao i Write data ulaz (za upisivanje podatka/rezultata ALU) su 32-bitni, pošto su sadržaju registara i rezultat sa izlazu ALU 32-bitni.

5.3.3 Dio datapath-a namijenjen kreiranju memory-reference instrukcija *lw* i *sw*

Da bi prišli implementaciji dijela datapath-a namijenjenog kreiranju memory-reference instrukcija *lw* i *sw*, sublimirajmo simbolički kod zapisivanja, tabela 1, i rezultate koji se postižu izvršavanjem ovih instrukcija, tabela 2. Simbolička forma zapisivanja instrukcija *lw* i *sw*, tabela 1,

lw/*sw* $x, Astart($y)

gdje je obilježje indeks registra y upisano u polju rs mašinskog koda instrukcija, obilježje registra x u polju rt mašinskog koda instrukcija, a Astart u address-nom polju mašinskog koda instrukcija, rezultira sljedećim ishodima, tabela 2:

Reg(rt)←Mem[Reg(rs)+sign_extend(address)] (kod instrukcije *lw*)

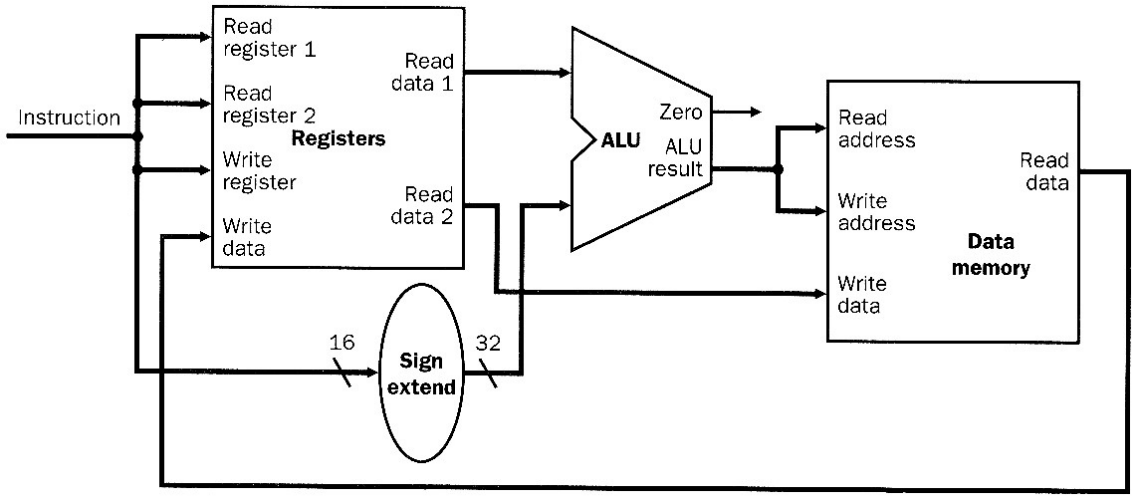
Mem[Reg(rs)+sign_extend(address)]←Reg(rt) (kod instrukcije *sw*)

gdje je sa sign_extend označeno kopiranje znaka 16-bitne adrese, zapisane u address-nom polju instrukcija, do 32-bitne dužine (pogledaj uvodne napomene poglavlja 4), sa Mem[Add] memorijska lokacija označena adresom Add, a sa Reg(rs) i Reg(rt) registari označeni poljima rs i rt instrukcija *lw* i *sw*.

Pojednostavljeno, operandi nad kojima ALU funkcioniše u slučaju instrukcija *lw*/*sw* nalaze se

– u registru označnom rs poljem instrukcija, i

– u address-nom polju instrukcije, pri čemu sadržaj address-nog mora biti produžen, kopiranjem znaka broja (sign_extend) do 32-bitne dužine prije pristupa na drugi ulaz ALU.



Slika 4. Dio datapath-a namijenjen kreiranju memory-reference instrukcija *lw* i *sw*.

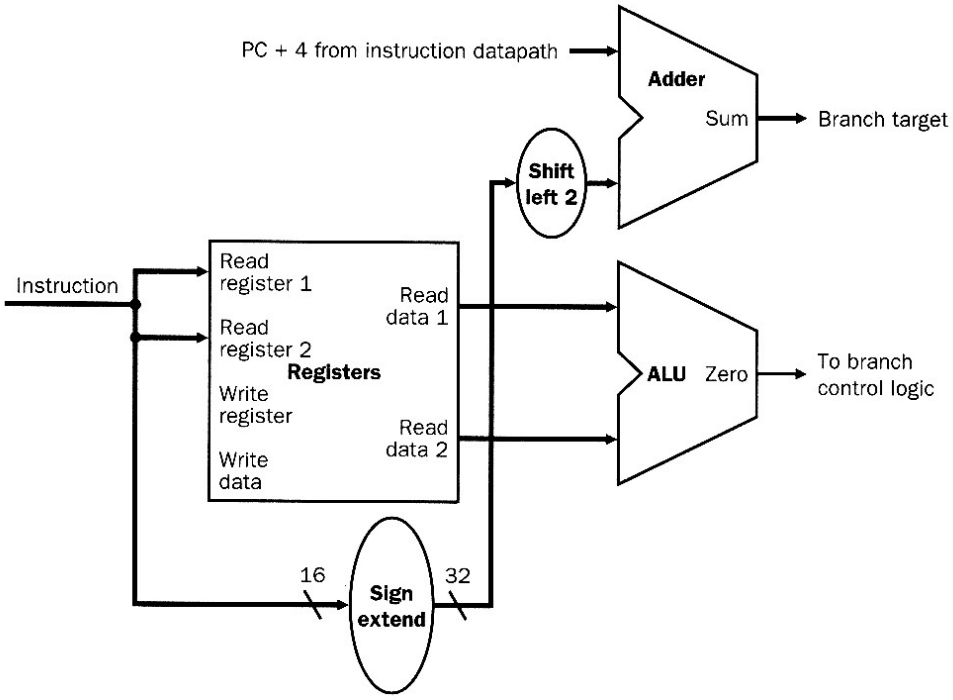
Shodno tome, u cilju implementiranja *lw*/*sw* instrukcija, na adresni ulaz Read register 1 Registers jedinice treba dovesti rs polje instrukcije, da bi se na Read register 1 izlazu iste jedinice dobio prvi operand nad kojim funkcioniše ALU. Na drugi ulaz ALU dovodi se 16-bitni sadržaj address-nog polja instrukcije, produžen, kopiranjem njegovog znaka, do 32-bitne dužine. ALU treba da izvrši sabiranje operanada i da svojim rezultatom adresira lokaciju Data memory

- sa koje će biti pročitan podatak koji će kasnije biti upisan u registar označen poljem rt u slučaju instrukcije *lw*, ili
- u koju će biti upisan podatak pročitan iz registra označenog poljem rt u slučaju instrukcije *sw*.

Razmotrimo najprije implementiranje instrukcije *lw*. Rezultat ALU adresira, na "Read address" ulazu, lokaciju Data memory koja na Read data izlazu daje podatak (sadržaj) sa adresirane lokacije koji treba upisati u registar označem poljem rt instrukcije. Shodno tome, sadržaj rt polja instrukcije treba dovesti na adresni Write register ulaz Registers jedinice, čime se adresira registar u koji će biti upisan podatak pročitan iz Data memory jedinice.

Razmotrimo sada implementaciju instrukcije *sw*. Rezultat ALU adresira, na "Write address" ulazu, lokaciju Data memory u koju treba upisati podatak pročitan iz registra označenog rt poljem instrukcije. Shodno tome, sadržaj rt polja instrukcije treba dovesti na adresni Read register 2 ulaz Registers jedinice, da bi se na Read data 2 izlazu iste jedinice dobio sadržaj adresiranog registra. Pročitani sadržaj registra dovodi se na Write data ulaz Data memory u cilju upisivanja u lokaciju adresiranu rezultatom ALU.

NAPOMENA 1: Primijetimo da se ista Registers jedinica (procesorski registri $0–$31) upotrebljava prilikom implementacija instrukcija R-tipa (sekcija 5.3.2) i prilikom implementacije instrukcija *lw*/*sw*. Medjutim, prilikom implementacije instrukcija R-tipa, registar u koji se upisuje rezultat ALU adresira se dovodjenjem **sadržaja rd polja instrukcije na Write register adresni ulaz Registers jedinice**, dok se **na Write data ulaz iste jedinice dovodi rezultat ALU (ALU result)**. Suprotno tome, prilikom implementacije instrukcioje *lw*, registar u koji se upisuje podatak, pročitan iz Data memory, adresira se dovodjenjem **sadržaja rt polja instrukcije na Write register adresni ulaz Registers jedinice**, dok se **na Write data ulaz iste jedinice dovodi podatak pročitan sa Read data izlaza Data memory**. Drugim riječima, ista Registers jedinica upotrebljava se za različite namjene prilikom implementacija instrukcija R-tipa i instrukcije *lw*. Da bi to bilo moguće, Write register adresni ulaz i Write data ulaz Registers jedinice treba share-ovati za različite namjene (ulaze) postavljanjem multipleksora na ovim ulazima, i to multipleksora čija veličina (broj ulaza) odgovara namjenama (ovdje instrukcijama R-tipa i instrukcije *lw* – dakle multipleksora sa po 2 ulaza). Medjutim, ovo je zadatak koji ćemo realizovati kasnije – prilikom kreiranja jedinstvenog datapath-a.



Slika 5. Dio datapath-a namijenjen kreiranju *beq* instrukcije.

NAPOMENA 2: Primijetimo da se ista ALU upotrebljava prilikom implementacija instrukcija R-tipa (sekcija 5.3.2) i prilikom implementacija instrukcija *lw*/*sw*. Prilikom implementacije instrukcija R-tipa, **na drugi ulaz ALU dovodi se sadržaj registra označenog poljem rt instrukcije**. Sa druge strane, prilikom izvršavanja instrukcija *lw* i *sw*, **na drugi ulaz ALU dovodi se 16-bitni sadržaj address-nog polja instrukcija, produžen, kopiranjem njegovog znaka, do 32-bitne dužine**. Drugim riječima, ista ALU upotrebljava se za različite namjene prilikom implementacija instrukcija R-tipa i instrukcija *lw*/*sw*. Da bi se to omogućilo, drugi ulaz ALU treba share-ovati za različite namjene (ulaze) postavljanjem multipleksora na ovom ulazu, i to multipleksora čija veličina (broj ulaza) odgovara namjenama (ovdje instrukcijama R-tipa i instrukcijama *lw*/*sw* – dakle multipleksora sa 2 ulaza). Ipak, ovo je zadatak koji će biti realizovan prilikom kreiranja jedinstvenog datapath-a.

5.3.4 Dio datapath-a namijenjen kreiranju instrukcije uslovnog skoka/grananja *beq*

Da bi prišli implementaciji dijela datapath-a namijenjenog kreiranju instrukcije uslovnog skoka/grananja *beq*, sublimirajmo simbolički kod zapisivanja, tabela 1, i rezultate koji se postižu izvršavanjem ove instrukcija, tabela 2. Simbolička forma zapisivanja instrukcije *beq*, tabela 1,

beq $x, $y, Offset

gdje su obilježja registara x i y redom upisana u polja rs i rt mašinskog koda instrukcije, a Offset u immediate polju mašinskog koda instrukcije, rezultira sljedećim ishodima, tabela 2:

– Izračunavanje ciljne adrese uslovnog skoka/grananja,

$$Target \leftarrow (PC+4)+sign\_extend(offset)<<2$$

– Ispitivanjem ispunjenosti uslova jednakosti operanada (sadržaja registara označenih poljima rs i rt instrukcije) i, ukoliko je uslov jednakosti zadovoljen, skokom na izračunatu ciljnu adresu grananja Target,

$$PC \leftarrow Target, \text{ ali samo ako je } Reg(rs)=Reg(rt)\ (Zero=1)$$

gdje je sa sign_extend označeno kopiranje znaka 16-bitne PC-relativne adrese (Offset-a), zapisane u immediate polju instrukcije, do 32-bitne dužine (pogledaj uvodne napomene poglavlja 4), sa <<2 shift-ovanje u lijevu stranu za 2 mjesta (množenje PC-relativne adrese sa

4), a sa Reg(rs) i Reg(rt) registri označeni poljima rs i rt instrukcije *beq*. Ukoliko uslov jednakosti nije zadovoljen, u PC registra se upisuje njegova inkrementirana vrijednost, PC←PC+4.

NAPOMENA: Prilikom izvršavanja instrukcija uslovnog i bezuslovnog skoka, buduća destinacija programa (adresa instrukcije koja sljedeća treba da se izvrši) mora biti adresa lokacije, a ne bilo kod drugog byte-a. Drugim riječima, u ovim slučajevima, adresa sljedeće instrukcije mora se završavati sa $00_{(2)}$. Pošto je činjenica da $00_{(2)}$ mora postojati u adresi sljedeće instrukcije, ove dvije nule se ne zapisuju u address-nim poljima mašinskog koda instrukcija uslovnog i bezuslovnog skoka (pogledaj napomenu 6 u poglavlju 3.7). Medjutim, prilikom implementacije instrukcija uslovnog i bezuslovnog skoka mora se zapisati $00_{(2)}$ na kraju adrese instrukcije koja sljedeća treba da se izvrši. To se obavlja shift-ovanjem/pomjeranjem, za 2 mjesta u lijevu stranu, sadržaja address-nih polja mašinskog koda instrukcija uslovnog i bezuslovnog skoka. Uz to, PC-relativna adresa zapisana u address-nom polju mašinskog koda instrukcije *beq*, uskladjena je sa adresom instrukcije koja u memoriji računara slijedi neposredno nakon instrukcije uslovnog skoka/grananja, odnosno uskladjena je sa PC+4 (pogledaj napomenu u poglavlju 3.7, i na strani 30).

Razmotrimo hardware-sku mplementaciju prethodnih navoda prikazanu na slici 5. 32-bitni sabirač Add izračunava ciljnu adresu grananja Target. Shodno tome, kao i navodima iz prethodne napomene, na njegove ulaze dovode se

- sadržaj PC registra inkrementiran za 4 (PC+4), koji je izračunat u dijelu datapath-a prezentiranom na slici 2 (nazvan "instruction datapath" na slici 5),
- 16-bitni sadržaj address-nog polja *beq* instrukcije produžen, kopiranjem znaka, do 32-bitne dužine i shift-ovan u lijevu stranu za 2 mjesta.

ALU ispituje ispunjenost uslova jednakosti operanada uzetih iz registara označenih poljima rs i rt instrukcije. Shodno tome, na Read registar 1 i Read reister 2 adresne ulaze Registers jedinice dovode se sadržaji polja rs i rt (kao kod instrukcija R-tipa). ALU na svom izlazu generuše Zero signal koji svojom jediničnom vrijednošću, Zero=1, signalizira ispunjenost uslova jednakosti operanada (za Zero=0, uslov jednakosti operanada nije zadovoljen, odnosno operandi su različiti).

Primijetimo sljedeće 2 činjenice:

1. Prilikom hardware-ske implementacije *beq* instrukcije, upotrebljava se ista Registers jedinica kao prilikom implementacije instrukcija R-tipa i instrukcija *lw*/*sw*, iako se u slučaju *beq* instrukcije ne vrši povratno upisivanje u Registers jedinicu (kao prilikom izvršavanja instrukcija R-tipa i *lw* instrukcije). Medjutim, o ovoj činjenici vodiće računa kontrolni signali Registers jedinice i kontrolnog interfejsa koji će biti postavljen na ulazima Registers jedinice (neophodnost upotrebe kontrolnog interfejsa kod Registers jedinice razmatrana je u napomeni iz sekcije 5.3.3).
2. Upisivanje ciljne adrese grananja Target u PC registar ukoliko je ispunjen uslov grananja (Zero=1) nije implementiran na slici 5, kao ni upisivanje inkrementiranog sadržaja PC registra (PC+4) ukoliko uslov grananja nije zadovoljen (Zero=0). Ovaj dio datapath-a biće implementiran prilikom kreiranja jednistvenog datapath-a i kontrolne jedinice.

5.3.5 Dio datapath-a namijenjen kreiranju instrukcije bezuslovnog skoka *j*

Instrukcija bezuslovnog skoka *j* implementira se zamjenom nižih 28 bitova tekućeg sadržaja PC registra (pogledaj napomenu 7 i konvenciju navedene u poglavlju 3.7) sa 26-bitnom adresom iz address-nog polja mašinskog koda instrukcije, tabela 1, shift-ovanom za 2 mjesta u lijevu stranu (pogledaj napomenu navedenu u sekciji 5.3.4). Najviša 4 bita PC registra, PC [31–28], zadržavaju svoj tekući sadržaj (pogledaj napomenu 7 i konvenciju iz poglavlja 3.7). Ipak, implementacija instrukcije *j* biće predstavljena tek prilikom dizajniranja potpunog datapath-a sa kontrolnom jedinicom.

Razmatrani djelovi datapath-a namijenjeni kreiranju pojedinačnih oblika instrukcija biće upotrijebljeni za krairanje jedinstvenog datapath-a u slučaju jednotaktne, ali i u slučaju višetaktne implementacije procesora.