

1) Aşağıda verilen komutlar Tek Saat Çevrimli MIPS Veri Yolu (Single Cycle Datapath) üzerinde gerçekleştirilebilmektedir.

```
lw_add rd, (rs), rt      # R[rd] = Memory[R[rs]] + R[rt];
addi_st (rs), rs, imm     # Memory[R[rs]] = R[rs] + imm;
sll_add rd, rs, rt, imm   # R[rd] = (R[rs] << imm) + R[rt];
```

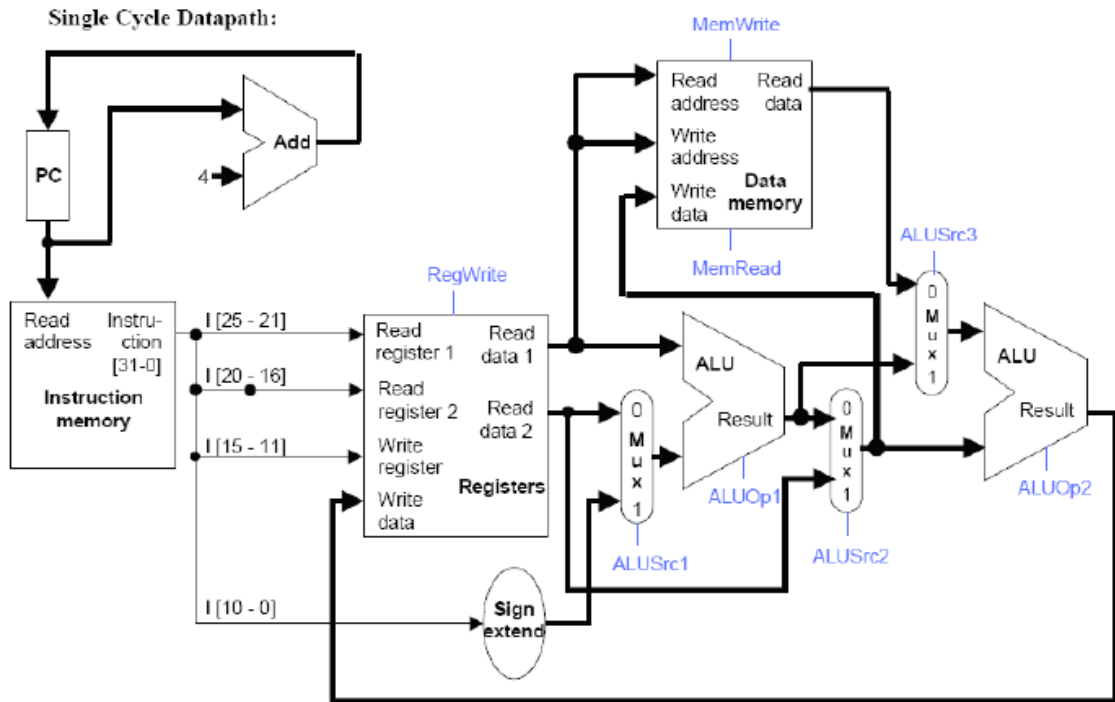
Bütün komutlar aşağıdaki komut formatını kullanmaktadır:

op (bits 31-26)	rs (bits 25-21)	rt (bits 20-16)	rd (bits 15-11)	imm (bits 10-0)
--------------------	--------------------	--------------------	--------------------	--------------------

Üstteki komutların doğru çalışabilmesi için üretilmesi gereken kontrol işaretlerini Tablodaki boşluklara doldurunuz?

ALUop Kontrol girişi için ADD, SUB, SLL, PASS_A veya PASS_B sembolik kodları kullanabilirsiniz. Burada PASS: ALU üzerinde operandın değişime uğramadan çıkışa transferi işlemini göstermektedir.

Inst	ALUSrc1	ALUSrc2	ALUSrc3	ALUOp1	ALUOp2	MemRead	MemWrite	RegWrite
lw_add								
addi_st								
sll_add								



2) a) MIPS Mimarisine aşağıdaki komut ilave edilmek istendiğini düşünün.

Sub3 rd, rs, rt, ru ; rd= ru - rs - rt

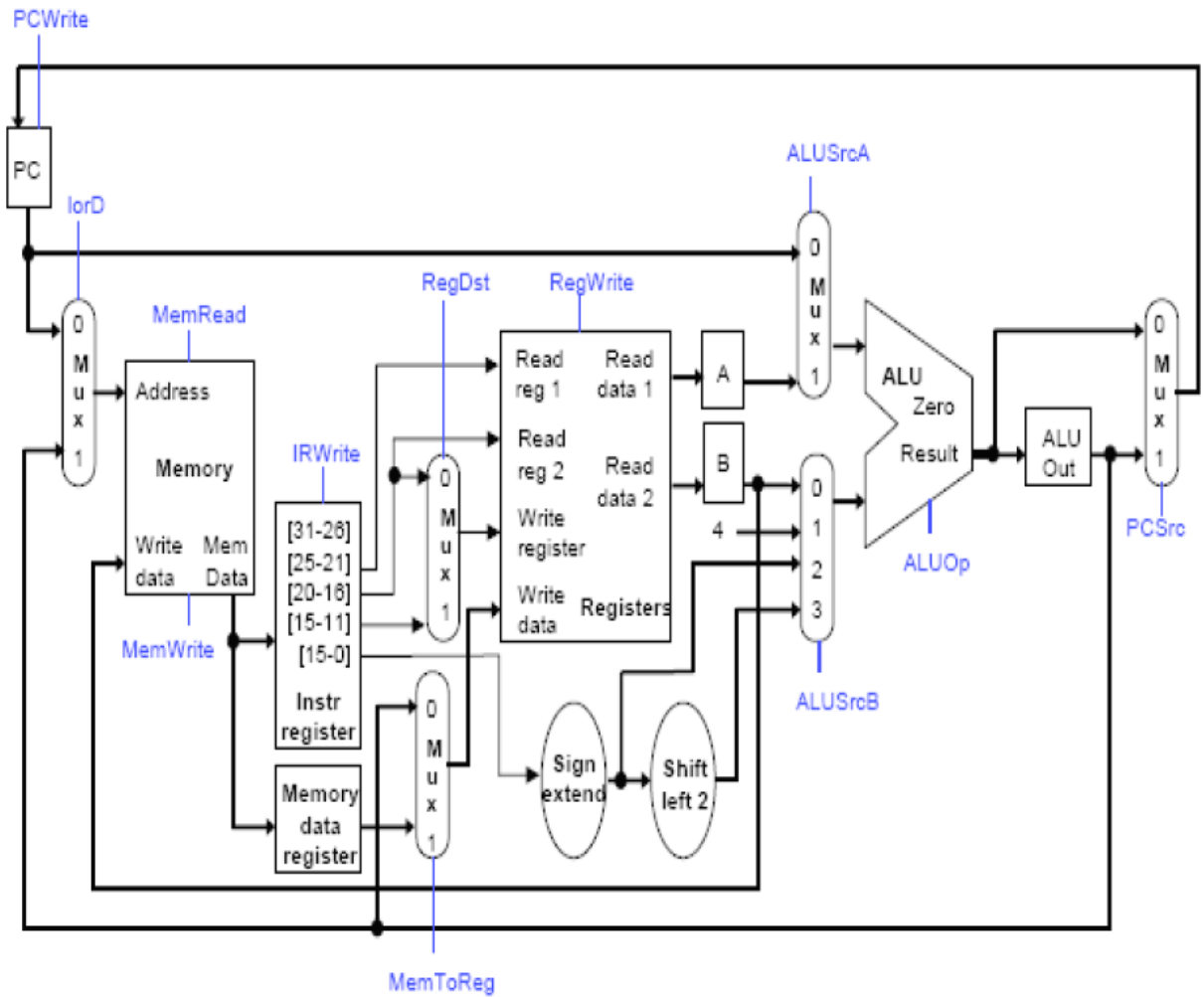
rd= ru – (rs + rt) olarak ta hesaplanabilir.

Aşağıdaki R türü komut formatı üzerindeki shamt değeri ru register tutmak için de kullanılabilir.

op (bits 31-26)	rs (bits 25-21)	rt (bits 20-16)	rd (bits 15-11)	shamt/ru (bits 10-6)	func (bits 5-0)
--------------------	--------------------	--------------------	--------------------	-------------------------	--------------------

Sub3 komutunu aşağıdaki Çok Çevrim MIPS Veri Yolunun desteklemesi için gereken değişimleri blok diyagram üzerinde gösteriniz?

Not: Temel uniteleri değiştirmeden sadece multiplexer ve bağlantı (wire) ilave ediniz.



b) Program 1 klasik add ve sub komutları, Program 2 ise Sub3 komutu (tek komut) kullanarak aynı işlemi yapmaktadırlar.

Program 2 , Program 1 'e göre %kaç hızlı icra edilmiş olur?

Not [a] şıkında görülen MIPS Veri Yolu üzerinde Program 1 ve Program 2 icrasının kaç saat çevrimi süreceğinden yararlanınız].

Program 1

```
lw    $t0, 0($a0)
lw    $t1, 4($a0)
lw    $t2, 8($a0)
add   $t3, $t2, $t1
sub   $t3, $t0, $t3
sw    $t3, 0($a1)
```

Program 2

```
lw    $t0, 0($a0)
lw    $t1, 4($a0)
lw    $t2, 8($a0)
sub3  $t3, $t2, $t1, $t0
sw    $t3, 0($a1)
```

3) MIPS Assembler Programı ile 12345670_{16} adresinden başlayarak 4 byte A datası ve 12345674_{16} adresinden başlayarak 4 byte B datası okunup 4 byte $|A-B|$ işleminin sonucu 12345678_{16} adresinden başlayarak yazdırılmak istenmektedir.

MIPS Assembler komutları ile programı yazınız?

4) Aşağıdaki MIPS Assembler program parçası kullanılarak \$s0 ve \$s1 içeriği ile ilgili yapılan değişikliği belirtiniz.?

(Ayrıca Komut yanlarına yaptığınız açıklama ile değişimleri de gösteriniz)

```
sll $t0,$s0,14
srl $t0,$t0,24
sll $t0,$t0,2
andi $s1,$s1,0xfe03
ori $s1,$s1,$t0
```

5) $A[10] = A[20] + 20$ dizin işlemini gerçekleyen aşağıdaki MIPS assembler program doğru çalışmamaktadır.

a) Programın yaptığı hatayı bulunuz

b) Programda gerekli değişikliği yaparak hatayı düzeltiniz?

```
lui    $s0, 16
addi   $s0, $zero, 32768
lw     $t0, 80($s0)
addi   $t0, $t0, 20
sw     $t0, 40($s0)
```

Dizin Taban adresi \$s0 register içinde saklanmaktadır (= $1081344_{10} = 108000_{16}$).

$32768_{10} = 8000_{16}$

6) MIPS mimarisinde komut çevrim adımları için işlem süreleri aşağıda verilmiştir.

IF = 7 ns, ID = 8 ns, EX = 15 ns, MA = 10 ns, WB = 8 ns

Pipeline registering 2 ns olsun...

- Tek çevrimli MIPS datapath çevrim süresi (cycle time) nedir?
- Çoklu çevrimli MIPS datapath çevrim süresi (cycle time) nedir?
- Pipelined datapath çevrim süresi (cycle time) nedir?
- Şıklarda hesapladığınız süreleri kullanarak ardarda 4 adet add komutunu (data bağımlılığı olmadığı kabulüyle) icra etmek için gereken süreleri i), ii) ve iii) için hesaplayınız?
 - Tek Çevrimli MIPS datapath kullanarak,
 - Çok Çevrimli MIPS datapath kullanarak,
 - Pipelined MIPS datapath kullanarak

7) A) Aşağıdaki MIPS Assembler programını pipelined MIPS mimarisi üzerinde çalıştırıldığında oluşan problemleri (hazards) belirtiniz, hangi satırlarda oluştuğunu gösteriniz?

lw \$t1, 0(\$t0)

lw \$t2, 4(\$t0)

add \$t3, \$t1, \$t2

sw \$t3, 12(\$t0)

lw \$t4, 8(\$t0)

add \$t5, \$t1, \$t4

sw \$t5, 16(\$t0)

B) Bu problemi gidermek için alınan önlemleri bir pipeline zamanlama diyagramı çizerek gösteriniz? Kaç saat çevriminde programın icra edildiğini yazınız?

C) Program komutlarının sırasını değiştirerek (sadece 1 komut) a. şıkkındaki pipeline hazard probleminin gecikme (stall) konulmadan giderilebileceğini gösteriniz ? (pipeline zamanlama diyagramı çizilebilir) Yeni durumda kaç saat çevriminde programın icra edildiğini yazınız?

8) a) Aşağıda verilen programın pipelined MIPS üzerinde icrasını **zamanlama diyagramı** üzerinde gösteriniz, programın icrası **kaç saat çevriminde** tamamlanır?

addi \$s0, \$s0, 20

lw \$s1, 0(\$s0)

add \$s2, \$s0, \$s1

add \$s3, \$s3, \$s0

add \$s3, \$s1, \$s4

b) Yukarıda verilen MIPS program önce single cycle MIPS ile sonra pipelined MIPS ile icra edilmiş olsun.. (Saat frekansı $T_c=100$ ps alınacak)

Single cycle MIPS e göre pipelined MIPS programın icrasında yapılan işlemleri ne kadar hızlandırır?

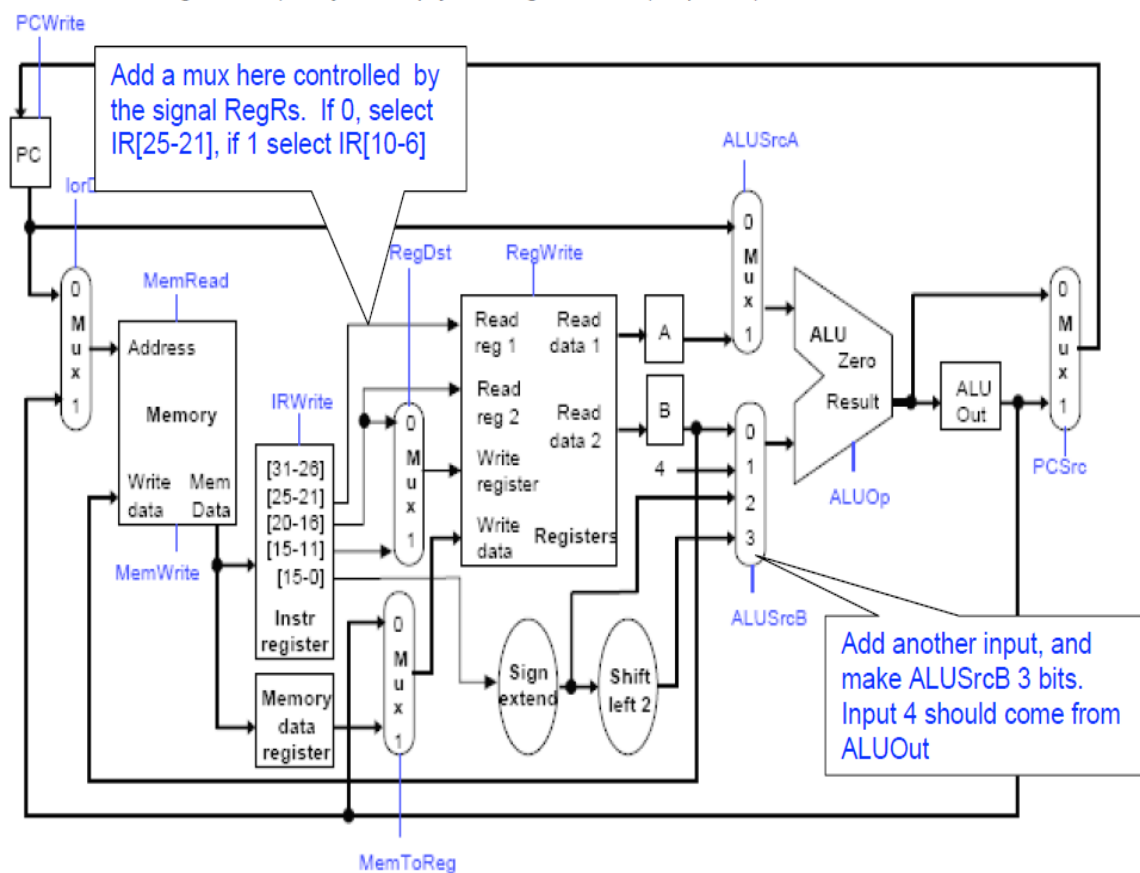
(**speed up faktörünü** hesaplayınız)

ÇÖZÜMLER

1-

Inst	ALUsrc1	ALUsrc2	ALUsrc3	ALUop1	ALUop2	MemRead	MemWrite	RegWrite
lw_add	X	1	0	X	ADD	1	0	1
addi_st	1	0	X	PASS_B	ADD	0	1	0
sll_add	1	1	1	SLL	ADD	0	0	1

2) a)



b)

Program 1 27 cycles
Program 2 24 cycles

Program 2 is faster by 3/27)

3)


**Program to calculate Absolute value of difference between
2 input numbers: $|A - B|$ (demonstrates if)**

Program reads A from 4 bytes of memory starting at address 12345670_{16} .

Program reads B from 4 bytes of memory starting at address 12345674_{16} .

Program writes $|A-B|$ to 4 bytes of memory starting at address 12345678_{16} .

<u>Assembler</u>	<u># Comment</u>
lui \$10, 0x1234	
ori \$10, \$10, 0x5670	# put address of A into register \$10
lw \$4, 0(\$10)	# read A from memory into register \$4
lw \$5, 4(\$10)	# read B from memory into register \$5 (A address+4)
sub \$12, \$5, \$4	# subtract A from B => B-A into register \$12
bgez \$12, +1	# branch if B-A is positive to 'sw' instruction
sub \$12, \$4, \$5	# subtract B from A => A-B into register \$12
sw \$12, 8(\$10)	# store register \$12 value, $ A-B $, into memory



4)

Write a sequence of no more than six MIPS instructions that extracts bits 17:11 of register \$s0 and inserts them into bits 8:2 of register \$s1, leaving all the remaining bits of \$s1 unchanged. You may use \$t registers as temporaries.

```
sll $t0,$s0,14      # turn bits 17:11 of $s0 into bits 8:2 of $t0
srl $t0,$t0,24
sll $t0,$t0,2
# everything else in $t0 should be 0
andi $s1,$s1,0xfe03 # zero out bits 8:2 in $s1
ori  $s1,$s1,$t0
```

5) a)

ANSWER

As a result of

```
lui    $s0, 16  
addi $s0, $zero, 32768
```

instructions, the content of `$s0` is 1015808 (1048576 - 32768), since sign extension in

```
addi $s0, $zero, 32768
```

turned 32768 into - 32768

b)

```
lui    $s0, 16  
ori    $s0, $zero, 32768  
lw     $t0, 80($s0)  
addi   $t0, $t0, 20  
sw     $t0, 40($s0)
```

6)

a) The time for the longest instruction, $lw = 7 + 8 + 15 + 10 + 8 = 48ns$

b) The time for the longest stage = 15ns

c) The time for the longest stage plus the overhead = $15 + 2 = 17ns$

d) i) single cycle $48 * 4 = 192ns$

ii) multi cycle There are 4 cycles for 1 add instruction, so $4 * (15 * 4) = 240ns$

iii) pipelined Five cycles for the first add, plus one more for each additional

$$add = (5 + 3) * 17 = 136ns$$

7)

a) Data hazard problem var.. Her 2 add komut satırında data bağımlılığı bulunmaktadır..

b) **Add komutu** satırlarında pipeline diyagramında STALL konulmalıdır..

Toplam 13 cc de program icra olur.

c) `lw $t1, 0($t0)`

`lw $t2, 4($t0)`

`lw $t4, 8($t0)`

`add $t3, $t1, $t2`

`sw $t3, 12($t0)`

`add $t5, $t1, $t4`

`sw $t5, 16($t0)`

Pipeline diyagramı çizilirse 11 cc de programın icra edildiği görülür.

8)

A) 13 cycle gerekiyor...

addi \$s0, \$s0, 20	IF	ID	EX	MEM	WB									
lw \$s1, 0(\$s0)		IF	ID	stall	stall	EX	MEM	WB						
add \$s2, \$s0, \$s1			IF	ID	stall	stall	stall	stall	EX	MEM	WB			
add \$s3, \$s3, \$s0				IF	ID	stall	stall	stall	stall	EX	MEM	WB		
add \$s3, \$s1, \$s4					IF	ID	stall	stall	stall	stall	EX	MEM	WB	

b) Single cycle 1 komut 5x100=500ps, program 5 komut x 500= 2500 ps

pipelined 13cycle x 100p = 1300 ps

speed up = 2500 / 1300 = 1.92

MIPS Instruction Set Summary (Subset)

Opcodes	Example Assembly	Semantics
add	add \$1, \$2, \$3	$\$1 = \$2 + \$3$
sub	sub \$1, \$2, \$3	$\$1 = \$2 - \$3$
add immediate	addi \$1, \$2, 100	$\$1 = \$2 + 100$
add unsigned	addu \$1, \$2, \$3	$\$1 = \$2 + \$3$
subtract unsigned	subu \$1, \$2, \$3	$\$1 = \$2 - \$3$
add imm. Unsigned	addiu \$1, \$2, 100	$\$1 = \$2 + 100$
multiply	mult \$2, \$3	hi, lo = $\$2 * \3
multiply unsigned	multu \$2, \$3	hi, lo = $\$2 * \3
divide	div \$2, \$3	lo = $\$2 / \3 , hi = $\$2 \bmod \3
divide unsigned	divu \$2, \$3	lo = $\$2 / \3 , hi = $\$2 \bmod \3
move from hi	mfhi \$1	$\$1 = \text{hi}$
move from low	mflo \$1	$\$1 = \text{lo}$
and	and \$1, \$2, \$3	$\$1 = \$2 \& \$3$
or	or \$1, \$2, \$3	$\$1 = \$2 \$3$
and immediate	andi \$1, \$2, 100	$\$1 = \$2 \& 100$
or immediate	ori \$1, \$2, 100	$\$1 = \$2 100$
shift left logical	sll \$1, \$2, 10	$\$1 = \$2 \ll 10$
shift right logical	srl \$1, \$2, 10	$\$1 = \$2 \gg 10$
load word	lw \$1, \$2(100)	$\$1 = \text{ReadMem32}(\$2 + 100)$
store word	sw \$1, \$2(100)	$\text{WriteMem32}(\$2 + 100, \$1)$
load halfword	lh \$1, \$2(100)	$\$1 = \text{SignExt}(\text{ReadMem16}(\$2 + 100))$
store halfword	sh \$1, \$2(100)	$\text{WriteMem16}(\$2 + 100, \$1)$
load byte	lb \$1, \$2(100)	$\$1 = \text{SignExt}(\text{ReadMem8}(\$2 + 100))$
store byte	sb \$1, \$2(100)	$\text{WriteMem8}(\$2 + 100, \$1)$
load upper immediate	lui \$1, 100	$\$1 = 100 \ll 16$
branch on equal	beq \$1, \$2, Label	if ($\$1 == \2) goto Label
branch on not equal	bne \$1, \$2, Label	if ($\$1 \neq \2) goto Label
set on less than	slt \$1, \$2, \$3	if ($\$2 < \3) $\$1 = 1$ else $\$1 = 0$
set on less than immediate	slti \$1, \$2, 100	if ($\$2 < 100$) $\$1 = 1$ else $\$1 = 0$
set on less than unsigned	sltu \$1, \$2, \$3	if ($\$2 < \3) $\$1 = 1$ else $\$1 = 0$
set on less than immediate	sltui \$1, \$2, 100	if ($\$2 < 100$) $\$1 = 1$ else $\$1 = 0$
jump	j Label	goto Label
jump register	jr \$31	goto \$31
jump and link	jal Label	$\$31 = \text{PC} + 4$; goto Label

EK MIP KOMUT FORMATLARI

Instruction Formats

- ❖ All instructions are 32-bit wide, Three instruction formats:
- ❖ **Register (R-Type)**
 - ❖ Register-to-register instructions
 - ❖ Op: operation code specifies the format of the instruction

Op⁶

Rs⁵

Rt⁵

Rd⁵

sa⁵

funct⁶

- ❖ **Immediate (I-Type)**
- ❖ 16-bit immediate constant is part in the instruction

Op⁶

Rs⁵

Rt⁵

immediate¹⁶

- ❖ **Jump (J-Type)**
- ❖ Used by jump instructions

Op⁶

immediate²⁶

MIPS Instruction Set Architecture
COE 301 – Computer Organization – KFU734
© Mohamed Mubawar – slide 9

Çok Çevrimli MIPS Komut İşleme Aşamaları

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR ← Memory[PC] PC ← PC + 4			
Instruction decode/register fetch	A ← Reg [IR[25:21]] B ← Reg [IR[20:16]] ALUOut ← PC + (sign-extend (IR[15:0]) << 2)			
Execution, address computation, branch/jump completion	ALUOut ← A op B	ALUOut ← A + sign-extend (IR[15:0])	if (A == B) PC ← ALUOut	PC ← {PC [31:28], (IR[25:0]), 2'b00}
Memory access or R-type completion	Reg [IR[15:11]] ← ALUOut	Load: MDR ← Memory[ALUOut] or Store: Memory [ALUOut] ← B		
Memory read completion		Load: Reg[IR[20:16]] ← MDR		

<div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">0 \$zero constant 0 (Hdware)</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">1 \$at reserved for assembler</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">2 \$v0 expression evaluation &</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">3 \$v1 function results</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">4 \$a0 arguments</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">5 \$a1</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">6 \$a2</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">7 \$a3</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">8 \$t0 temporary: caller saves</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">... (callee can clobber)</div> <div style="border: 1px solid black; padding: 2px;">15 \$t7</div>	<div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">16 \$s0 callee saves</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">... (caller can clobber)</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">23 \$s7</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">24 \$t8 temporary (cont'd)</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">25 \$t9</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">26 \$k0 reserved for OS kernel</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">27 \$k1</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">28 \$gp pointer to global area</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">29 \$sp stack pointer</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 2px;">30 \$fp frame pointer</div> <div style="border: 1px solid black; padding: 2px;">31 \$ra return address (Hdware)</div>
---	--