UYGULAMA-1

Bilgisayar Mimarisi

Instruction Formats

R-type (6-bit opcode, 5-bit rs, 5-bit rt, 5-bit rd, 5-bit shamt, 6-bit function code)

31-26	25-21	20-16	15-11	10-6	5-0
opcode	*rs*	*rt*	*rd*	shamt	function

– I-type (6-bit opcode, 5-bit rs, 5-bit rt, 16-bit immediate)

31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*imm*

– J-type (6-bit opcode, 26-bit pseudo-direct address)

31-26	25-0
opcode	*pseudodirect jump address*

Mnemonic	Meaning	Type	Opcode	Funct
add	Add	R	0x00	0x20
addi	Add Immediate	I	0x08	NA
addiu	Add Unsigned Immediate	I	0x09	NA
addu	Add Unsigned	R	0x00	0x21
and	Bitwise AND	R	0x00	0x24
andi	Bitwise AND Immediate	I	0x0C	NA
beq	Branch if Equal	I	0x04	NA
bne	Branch if Not Equal	I	0x05	NA
div	Divide	R	0x00	0x1A
divu	Unsigned Divide	R	0x00	0x1B
j	Jump to Address	J	0x02	NA
jal	Jump and Link	J	0x03	NA
jr	Jump to Address in Register	R	0x00	0x08
jalr	Jump to Address in Register and Link	R	0x00	0x09

lbu	**Load Byte Unsigned**	**I**	**0x24**	**NA**
lhu	Load Halfword Unsigned	I	0x25	NA
lui	Load Upper Immediate	I	0x0F	NA
lw	Load Word	I	0x23	NA
mfhi	Move from HI Register	R	0x00	0x10
mflo	Move from LO Register	R	0x00	0x12
mfc0	Move from Coprocessor 0	R	0x10	NA
mult	Multiply	R	0x00	0x18
multu	Unsigned Multiply	R	0x00	0x19
nor	Bitwise NOR (NOT-OR)	R	0x00	0x27
xor	Bitwise XOR (Exclusive-OR)	R	0x00	0x26
or	Bitwise OR	R	0x00	0x25
ori	Bitwise OR Immediate	I	0x0D	NA
sb	Store Byte	I	0x28	NA

`sh`	Store Halfword	I	0x29	NA
`slt`	Set to 1 if Less Than	R	0x00	0x2A
`slti`	Set to 1 if Less Than Immediate	I	0x0A	NA
`sltiu`	Set to 1 if Less Than Unsigned Immediate	I	0x0B	NA
`sltu`	Set to 1 if Less Than Unsigned	R	0x00	0x2B
`sll`	Logical Shift Left	R	0x00	0x00
`srl`	Logical Shift Right (0-extended)	R	0x00	0x02
`sra`	Arithmetic Shift Right (sign-extended)	R	0x00	0x03
`sub`	Subtract	R	0x00	0x22
`subu`	Unsigned Subtract	R	0x00	0x23
`sw`	Store Word	I	0x08	NA

Jr

- The **jr** instruction loads the PC register with a value stored in a register.

As such, the jr Instruction can be called as such:

jr $t0

assuming the target jump location is located in $t0.

Jalr

The same as the **jr** instruction, except that the return address is loaded into the $ra register.

JAL

Jump and Link instructions are similar to the jump instructions, except that they store the address of the next instruction (the one immediately after the jump) in the return address ($ra; $31) register. This allows a subroutine to return to the main body routine after completion.

Example:
Let's say that we have a subroutine that starts with the label MySub. We can call the subroutine using the following line:

jal MySub ...

And we can define MySub as follows to return to the main body of the parent routine:

SORU 1

: Aşağıdaki komutların makine kodu karşılıklarını bulunuz?

- ADD $2, $3, $4
  - R-type A/L/S/C instruction
  - Opcode is 0’s, rd=2, rs=3, rt=4, func=000010
  - 000000 00011 00100 00010 00000 000010
- JALR $3
  - R-type jump instruction
  - Opcode is 0’s, rs=3, rt=0, rd=31 (by default), func=001001
  - 000000 00011 00000 11111 00000 001001
- ADDI $2, $3, 12
  - I-type A/L/S/C instruction
  - Opcode is 001000, rs=3, rt=2, imm=12
  - 001000 00011 00010 0000000000001100

- BEQ $3, $4, 4
  - I-type conditional branch instruction
  - Opcode is 000100, rs=00011, rt=00100, imm=4 (skips next 4 instructions)
  - 000100 00011 00100 0000000000000100

- SW $2, 128($3)
  - I-type memory address instruction
  - Opcode is 101011, rs=00011, rt=00010, imm=0000000010000000
  - 101011 00011 00010 0000000010000000

- J 128
  - J-type pseudodirect jump instruction
  - Opcode is 000010, 26-bit pseudodirect address is 128/4 = 32
  - 000010 00000000000000000000100000

SORU 2 :
Aşağıdaki pseudo komutların gerçek makine komutları cinsinden karşılığını yazınız?

(a)li $s0, 17 assembler pseudo-instruction

Actual machine instruction

addiu $s0, $zero, 17 (addi, ori or other equivalent answers also ok)

(b) bge $s0, $t0, there assembler pseudo-instruction

Actual machine instruction
slt $at, $s0, $t0 # $at = 1 if <, or 0 if ≥
beq $at, $zero, there

SORU 3 :

Suppose we have a 32-bit MIPS word containing the value 0x008A1021.
We would like to know what MIPS machine instruction this represents.

(a) Write this instruction word in binary.

0 0 0 0 0 0	0 0 1 0 0	0 1 0 1 0	0 0 0 1 0	0 0 0 0 0	1 0 0 0 0 1
op	rs	rt	rd	shamt	funct

(b) What is the format of this instruction? (circle)

R I J

c) Translate this instruction to assembly language. Use symbolic register names like
$t8 instead of absolute register numbers like $24.

addu $v0, $a0, $t2

SORU 4 :

Suppose we execute the following MIPS instructions

```
li $t0, 2
li $t1, 5
slt $t2, $t1, $t0
beq $t2, $zero, skip
addi $t1, $t2, 3
 skip:
 li $v0, 42
```

In the table, write down each of the registers changed during execution and their values after the code has executed.

Register	\$t0	\$t1	\$t2	\$v0
Value	2	5	0	42

SORU 5 :
DİZİN ELEMANLARINI SABİTLE TOPLAYIP BAŞKA BİR *DİZİNE* ATAMA PROGRAMINI MIPS KOMUTLARIYLA YAZINIZ?

```
for (i=0;i<n;i++) a[i]=b[i]+10;

        xor $2,$2,$2   # zero out index register (i)
        lw $3,n        # load iteration limit
        sll $3,$3,2    # multiply by 4 (words)
        li $4,a        # get address of a (assume < 2^16)
        li $5,b        # get address of b (assume < 2^16)
loop:   add $6,$5,$2   # compute address of b[i]
        lw $7,0($6)    # load b[i]
        addi $7,$7,10  # compute b[i]=b[i]+10
        add $6,$4,$2   # compute address of a[i]
        sw $7,0($6)    # store into a[i]
        addi $2,$2,4   # increment i
        blt $2,$3,loop # loop if post-test succeeds
```

SORU 6 Write a minimal sequence of MIPS instructions that accomplishes the following:

```
a[15] = a[14] - 15;
```

where `a` is an array of words already declared and initialized in `.data`.

```
li   $t0, a
lw   $t1, 56($t0)
addi $t1, $t1, -15
sw   $t1, 60($t0)
```

or the slightly longer:

```
addi $t0, $0, 56
lw   $t1, a($t0)
addi $t1, $t1, -15
addi $t0, $0, 60
sw   $t1, a($t0)
```

SORU 7 Write a sequence of MIPS instructions to branch to address `0xFFFFFFFC` if \$t1 is equal to \$t2. Recall that the destination of `j` or any branch instruction must be a label, not a constant

```
        beq $t1, $t2, long
        j fin
long:   lui  $t0, 0xffff
        addi  $t0, $t0, 0xfffc
        jr   $t0
fin:
```

SORU 8 Write the machine language encoding of the `beq` instruction in the following fragment. Express your answer in binary.

```
      addi $t0, $t3, 143
      sw   $s0, 165($t1)
      beq  $s4, $sp, skip
      addi $s0, $s0, 4
skip: sub  $t0, $t3, $s0
```

000100 10100 11101 0000 0000 0000 0001

SORU 9 Write a minimal sequence of MIPS instructions that accomplishes the following:

$$\$t3 = 35 * (\$t1 / \$t2) + 14$$

You can assume $t2 is not zero and that all integers are signed and small enough that the results of all operations do fit in 32 bits.

```
div  $t1, $t2
mflo $t0
addi $t3, $0, 35
mult $t3, $t0
mflo $t3
addi $t3, $t3, 14
```

SORU 10 Write a minimal sequence of MIPS instructions that makes the least-significant bit in $a0 the same as its most-significant bit without changing any other bit in it.

```
srl  $t0, $a0, 31
lui  $t2, 0xffff
ori  $t2, $t2, 0xfffe
   and  $t1, $a0, $t2
   or   $a0, $t1, $t0
```

also

```
srl  $t0, $a0, 31
srl $a0, $a0, 1
sll $a0, $a0, 1
or  $a0, $ao, $t0
```

SORU 11 —

Aşağıdaki MIPS programının amacını komutlara açıklama (comments) yazarak açıklayınız?

a0 registerinin başlangıçta n tamsayı değerine sahip oldugunu kabul ediniz..

```
begin:    addi $t0, $zero, 0
          addi $t1, $zero, 1
loop:     slt  $t2, $a0, $t1
          bne  $t2, $zero, finish
          add  $t0, $t0, $t1
          addi $t1, $t1, 2
          j    loop
finish:   add  $v0, $t0, $zero
```

YANIT 1:

```
begin:   addi $t0, $zero, 0        # $t0 = sum = 0
         addi $t1, $zero, 1        # $t1 = i = 1
loop:    slt  $t2, $a0, $t1        # (n<i)? or (i>n)?
         bne  $t2, $zero, finish   # exit loop if (i>n)
         add  $t0, $t0, $t1        # sum = sum + i
         addi $t1, $t1, 2          # i = i + 2
         j    loop                 # repeat loop
finish:  add  $v0, $t0, $zero      # result = sum
```

V0 Toplam sonucunu tutar.

İşlem: 1+3+5+... tek pozitif sayıların toplamını bulur (n < = olan tek sayıların toplamı)

SORU 12--

Aşağıda görülen C koda karşı gelen MIPS assembler programında hangi satırlarda hangi hataların yapıldığını açıklayınız?

C Code	MIPS Code	Line
`int fact (int n)`	`fact:`	1
`{`	`addi $sp, $sp, -2`	2
`if (n < 1) return f;`	`sw 1($sp), $at`	3
`else return n * fact(n - 1);`	`sw 0($sp), $a0`	4
`}`	`slti $t0, $a0, 1`	5
	`beq $t0, $zero, L1`	6
	`addi $v0, $zero, 1`	7
	`addi $sp, $sp, 8`	8
	`j $at`	9
	`L1: addi $a0, $a0, -1`	10
	`jal fact`	11
	`lw $a0, 0($sp)`	12
	`lw $at, 1($sp)`	13
	`addi $sp, $sp, 2`	14
	`mul $v0, $a0, $v0`	
	`j $at`	

Error 1:

In Lines 2 and 12, the stack should be increased by 8 bytes, not 2 bytes.

Error 2:

In Lines 3 and 4, the `sw`'s operands should be swapped.

Error 3:

In Lines 3, 8, 11, and 14, should use the `$ra` register, not `$at`.

Error 4:

In Lines 3 and 11 the displacement should be 4, not 1.

Error 5:

In Lines 8 and 14 should use `jr`, not `j`.

Soru-13

Aşağıdaki matematiksel ifadeyi hesaplayan bir MIPS assembler programı yazınız?

$$x^3 + 6x^2y + 12xy^2 + 8y^3$$

x değişkeni \$s0, y değişkeni \$s1, sonuç \$s2 saklayıcılarında tutulmaktadır..

Not: İşlemi basit yapmak için cebirsel manüplasyon kullanılabilir…

Solution *The easy way: calculate* $(x + 2y)^3$

```
sll $t0, $s1, 1    # $t0 = 2y
add $t0, $t0, $s0  # $t0 = x + 2y
mul $s2, $t0, $t0  # $s2 = (x + 2y)^2
mul $s2, $s2, $t0  # $s2 = (x + 2y)^3
```

Aşağıdaki MIPS Asembler Kodu icra edildiğinde;
slt, beq, addi ve j komutlarının makine kodu karşılıklarını veriniz?

Address	**MIPS assembler**		
0040 0020	**main:**	**slt**	**\$t0, \$a0, \$a1**
			beq \$t0, \$zero, exit
			addi \$a1, \$a1, 4
			j main
		exit:	**. . .**

Machine Code:

slt	0	4	5	8	0	42
beq	4	8	0	2		
addi	8	5	5	4		
j	2	0000 0100 0000 0000 0000 0010 00_2				

SORU 14 —

Aşağıdaki MIPS programının amacını komutlara açıklama (comments) yazarak açıklayınız?

a0 registerinin başlangıçta n tamsayı değerine sahip oldugunu kabul ediniz..

```
begin:   addi $t0, $zero, 0
         addi $t1, $zero, 1
loop:    slt  $t2, $a0, $t1
         bne  $t2, $zero, finish
         add  $t0, $t0, $t1
         addi $t1, $t1, 2
         j    loop
finish:  add  $v0, $t0, $zero
```

YANIT 1:

```
begin:    addi $t0, $zero, 0       # $t0 = sum = 0
          addi $t1, $zero, 1       # $t1 = i = 1
loop:     slt  $t2, $a0, $t1       # (n<i)? or (i>n)?
          bne  $t2, $zero, finish  # exit loop if (i>n)
          add  $t0, $t0, $t1       # sum = sum + i
          addi $t1, $t1, 2         # i = i + 2
          j    loop                # repeat loop
finish:   add  $v0, $t0, $zero     # result = sum
```

V0 Toplam sonucunu tutar.

İşlem: 1+3+5+... tek pozitif sayıların toplamını bulur (n < = olan tek sayıların toplamı)

Soru 15

Write the MIPS method that receives in $a0 a positive integer X and returns in $v0 the largest integer M that satisfies the following:

$$1 + 4 + 9 + 16 + \ldots + M^2 <= X$$

In other words, M is the largest integer such that the sum of the squares of the first M integers is not more than X. For example, if X=79 then M would be 5 because the sum 1+4+9+16+25 is 55, and had we added one more term (36), the sum (91) would have exceeded X. You can assume that the integer in $a0 is positive, and that all integers are signed and small enough that the individual squares and the final sum do fit in 32 bits.

```
        add  $s0, $0, $0        # s0 = sum
        add  $v0, $0, $0        # v0 = increasing int
loop:   addi $v0, $v0, 1        # v0++
        mult $v0, $v0
        mflo $t0                # t0 = v0 squared
        add  $s0, $s0, $t0      # update the sum
        slt  $t0, $s0, $a0
        bne  $t0, $0, loop      # repeat if sum < a0
        beq  $s0, $a0, done     # done if sum = a0
        addi $v0, $v0, -1       # backtrack if sum > a0
        done:   jr   $ra        # return to caller
```

SORU 16

MIPS Komut Setindeki tüm load ve store komutlarının formatını değiştirmek istediğimizi düşünün.

load ve store komutları R türü komutlar haline dönüştürülmek isteniyor. Offset (0) olarak alınacak yani 'immediate' operand gerekmeyecektir.
Yani bellek adresi hesaplamasında ALU ya ihtiyaç duyulmaması öngörülmüştür.

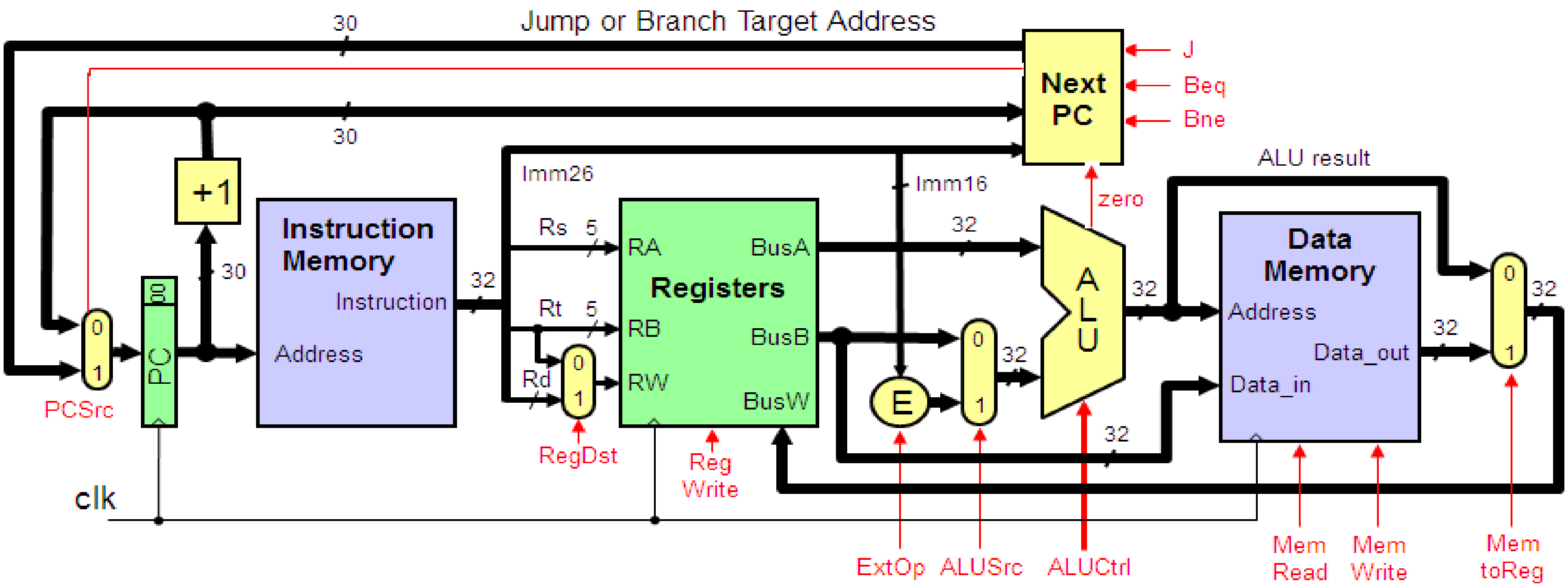
Yeni load ve store komutları formatı :

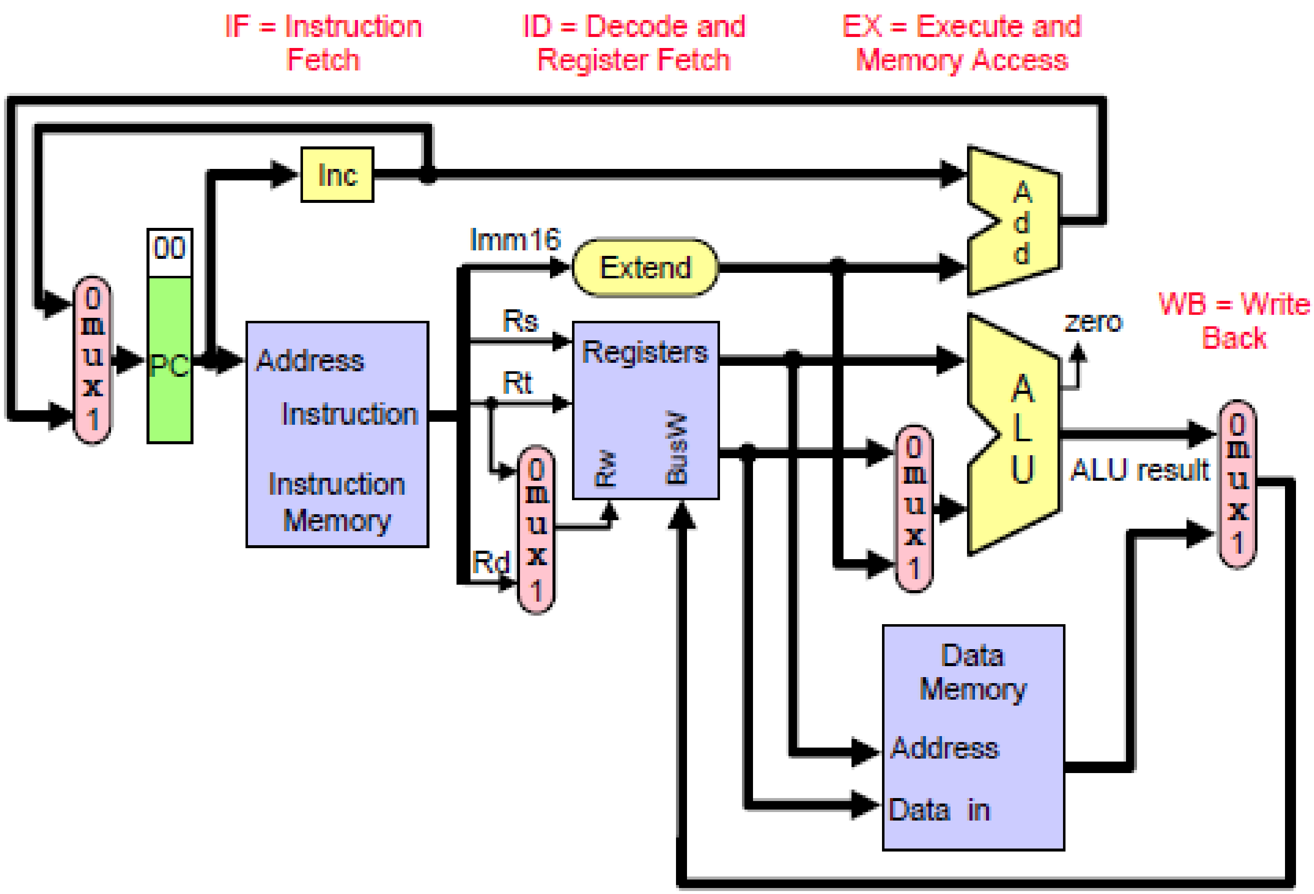
LW Rt, (Rs)
SW Rt, (Rs)

Burada Rs bellek adresini içeren register'i göstermektedir.

.

Single cycle datapath üzerinde gerekli donanımsal değişimi gösteriniz..?




IF = Instruction Fetch
ID = Decode and Register Fetch
EX = Execute and Memory Access
WB = Write Back
Inc
00
PC
0 mux 1
Address
Instruction
Instruction Memory
Imm16
Rs
Rt
Rd
Extend
Registers
Rw
BusW
Add
ALU
zero
ALU result
Data Memory
Address
Data in