EECE 417 Computer Systems Architecture

Department of Electrical and Computer Engineering
Howard University

Charles Kim

Spring 2007

Computer Organization and Design (3rd Ed)

-The Hardware/Software Interface

by

David A. Patterson
John L. Hennessy

Chapter 2

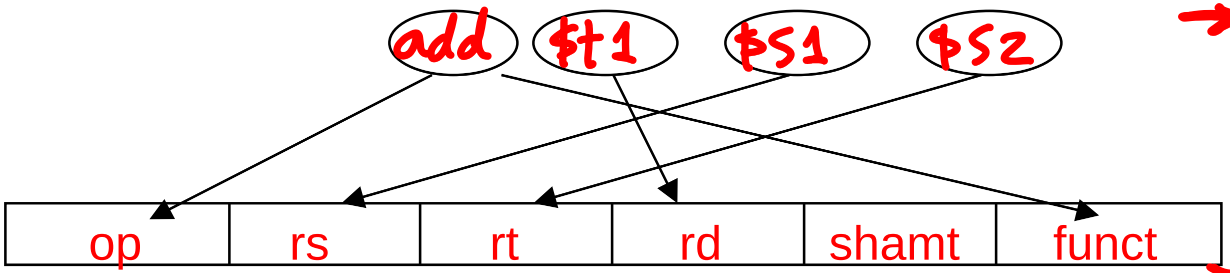
Instructions: Language of the Computer

Machine Language

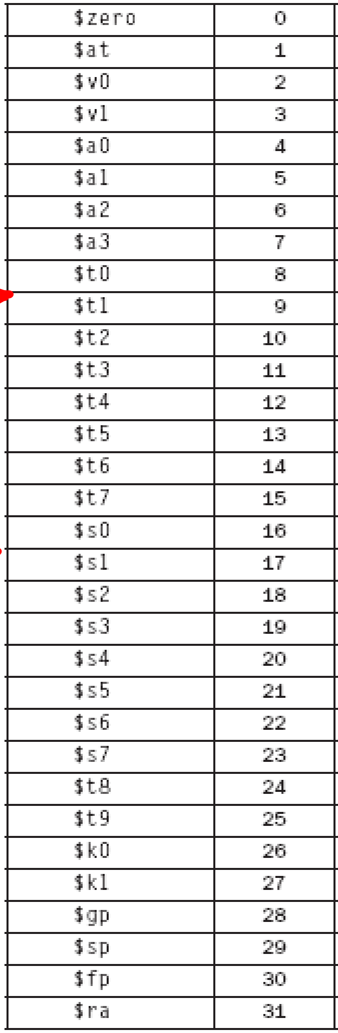
- Instructions, like registers and words of data, are 32 bits lo
- Arithmetic Instruction Format (R format):

```
add $t1, $s1, $s2
```

```
#  registers have numbers, $t1=9,  $s1=17,  $s2=18
```



op	6-bits	opcode that specifies the operation
rs	5-bits	register file address of the first source operand
rt	5-bits	register file address of the second source operan
rd	5-bits	register file address of the result's destination
shamt	5-bits	shift amount (for shift instructions)
funct	6-bits	function code augmenting the opcode



$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Language

- Instructions, like registers and words of data, are also 32 bits long
  - Example: `add $t1, $s1, $s2`
  - registers have numbers, `$t1=9, $s1=17, $s2=18`

- Instruction Format (R):

000000	10001	10010	01000	00000	100000
op	rs	rt	rd	shamt	funct

- *Can you guess what the field names stand for?*

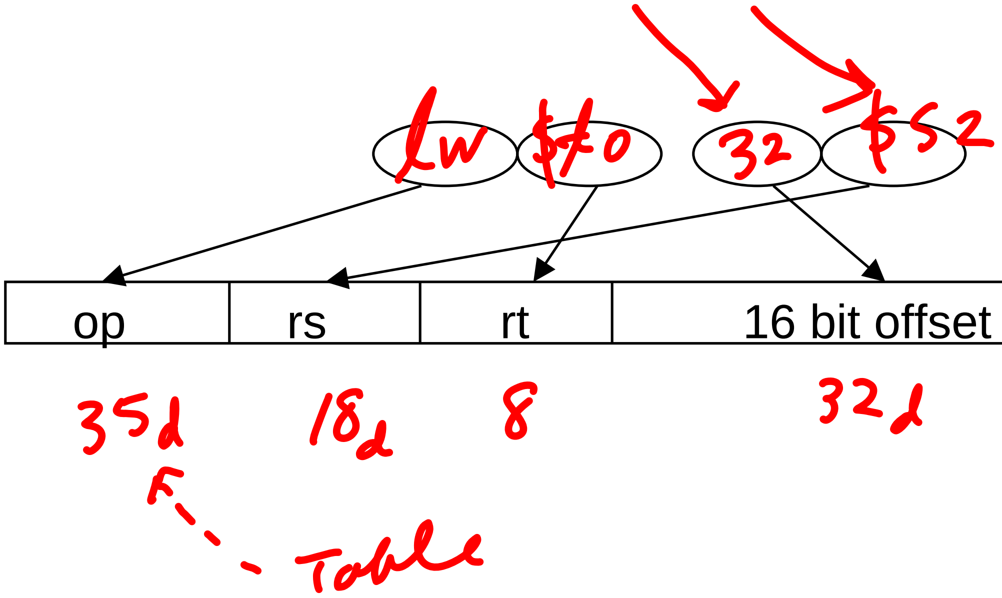
Need
Table for these

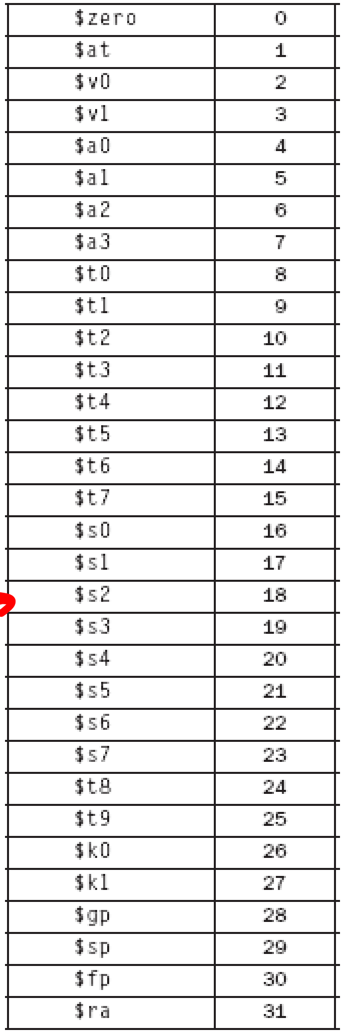
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Language

- Consider the load-word and store-word instructions,
- Introduce a new type of instruction format
  - I-type for data transfer instructions
- Load/Store Instruction Format (I format): Example

`lw $t0, 32($s2)`





$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

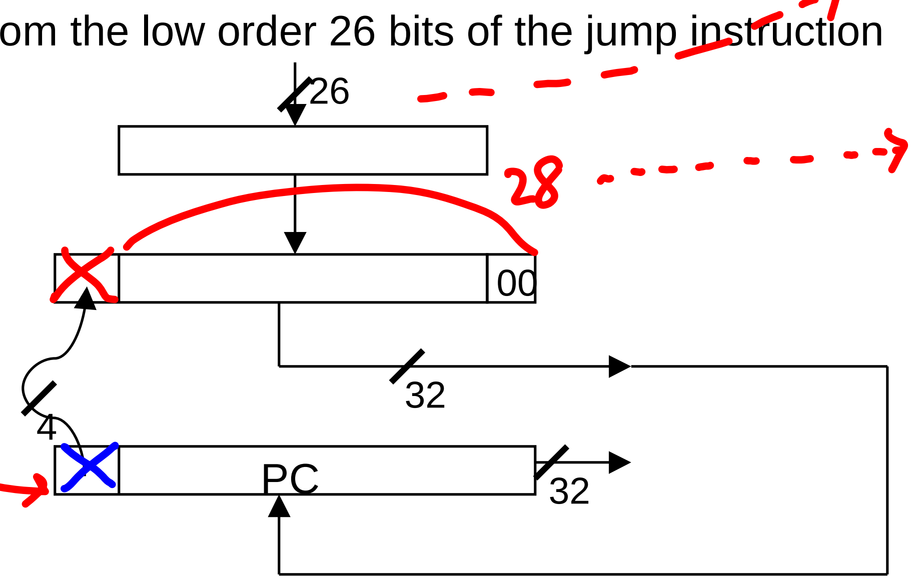
Control Flow Instructions (J Format)

- **Ubranch instruction or jump instruction:**

```
j   label               #go to label
```

- **Instruction Format (J Format):**

op	26-bit address



→ 2^{26} word = 64M word = 256 Mbyte

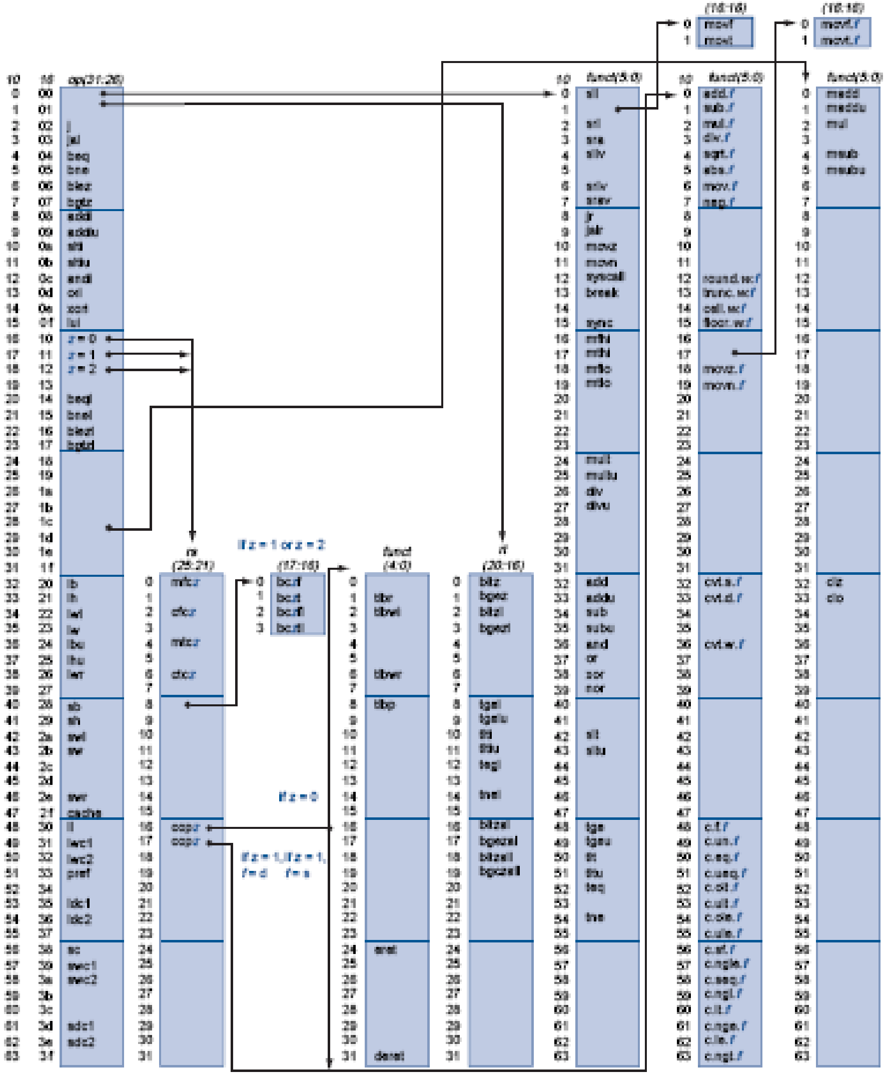
28

The upper 4 bits of PC unchanged

then, how do we jump the full range?

jr $ra

MIPS Opcode map



10	16	op(31:26)	10	funct(5:0)	10	funct(5:0)		funct(5:0)
0	00		0	sll	0	add.*f*	0	madd
1	01		1		1	sub.*f*	1	maddu
2	02	j	2	srl	2	mul.*f*	2	mul
3	03	jal	3	sra	3	div.*f*	3	
4	04	beq	4	sllv	4	sqrt.*f*	4	msub
5	05	bne	5		5	abs.*f*	5	msubu
6	06	blez	6	srlv	6	mov.*f*	6	
7	07	bgtz	7	srav	7	neg.*f*	7	
8	08	addi	8	jr	8		8	
9	09	addiu	9	jalr	9		9	
10	0a	slti	10	movz	10		10	
11	0b	sltiu	11	movn	11		11	
12	0c	andi	12	syscall	12	round.w.*f*	12	
13	0d	ori	13	break	13	trunc.w.*f*	13	
14	0e	xori	14		14	ceil.w.*f*	14	
15	0f	lui	15	sync	15	floor.w.*f*	15	
16	10	*z* = 0	16	mfhi	16		16	
17	11	*z* = 1	17	mthi	17		17	
18	12	*z* = 2	18	mflo	18	movz.*f*	18	
19	13		19	mtlo	19	movn.*f*	19	
20	14	beql	20		20		20	
21	15	bnel	21		21		21	
22	16	blezl	22		22		22	
23	17	bgtzl	23		23		23	
24	18		24	mult	24		24	
25	19		25	multu	25		25	
26	1a		26	div	26		26	
27	1b		27	divu	27		27	
28	1c		28		28		28	
29	1d		29		29		29	
30	1e		30		30		30	
31	1f		31		31		31	
32	20	lb	32	add	32	cvt.s.*f*	32	clz
33	21	lh	33	addu	33	cvt.d.*f*	33	clo
34	22	lwl	34	sub	34		34	
35	23	lw	35	subu	35		35	
36	24	lbu	36	and	36	cvt.w.*f*	36	
37	25	lhu	37	or	37		37	
38	26	lwr	38	xor	38		38	
39	27		39	nor	39		39	
40	28	sb	40		40		40	
41	29	sh	41		41		41	
42	2a	swl	42	slt	42		42	
43	2b	sw	43	sltu	43		43	
44	2c		44		44		44	
45	2d		45		45		45	
46	2e	swr	46		46		46	
47	2f	cache	47		47		47	
48	30	ll	48	tge	48	c.f.*f*	48	
49	31	lwc1	49	tgeu	49	c.un.*f*	49	
50	32	lwc2	50	tlt	50	c.eq.*f*	50	
51	33	pref	51	tltu	51	c.ueq.*f*	51	
52	34		52	teq	52	c.olt.*f*	52	
53	35	ldc1	53		53	c.ult.*f*	53	
54	36	ldc2	54	tne	54	c.ole.*f*	54	
55	37		55		55	c.ule.*f*	55	
56	38	sc	56		56	c.sf.*f*	56	
57	39	swc1	57		57	c.ngle.*f*	57	
58	3a	swc2	58		58	c.seq.*f*	58	
59	3b		59		59	c.ngl.*f*	59	
60	3c		60		60	c.lt.*f*	60	
61	3d	sdc1	61		61	c.nge.*f*	61	
62	3e	sdc2	62		62	c.le.*f*	62	
63	3f		63		63	c.ngt.*f*	63	

	(16:16)
0	movf
1	movt

	(16:16)
0	movf.*f*
1	movt.*f*

	rs (25:21)		funct (4:0)		rt (20:16)
0	mfc*z*	0		0	bltz
1		1	tlbr	1	bgez
2	cfc*z*	2	tlbwi	2	bltzl
3		3		3	bgezl
4	mtc*z*	4		4	
5		5		5	
6	ctc*z*	6	tlbwr	6	
7		7		7	
8		8	tlbp	8	tgei
9		9		9	tgeiu
10		10		10	tlti
11		11		11	tltiu
12		12		12	teqi
13		13		13	
14		14		14	tnei
15		15		15	
16	cop*z*	16		16	bltzal
17	cop*z*	17		17	bgezal
18		18		18	bltzall
19		19		19	bgezall
20		20		20	
21		21		21	
22		22		22	
23		23		23	
24		24	eret	24	
25		25		25	
26		26		26	
27		27		27	
28		28		28	
29		29		29	
30		30		30	
31		31	deret	31	

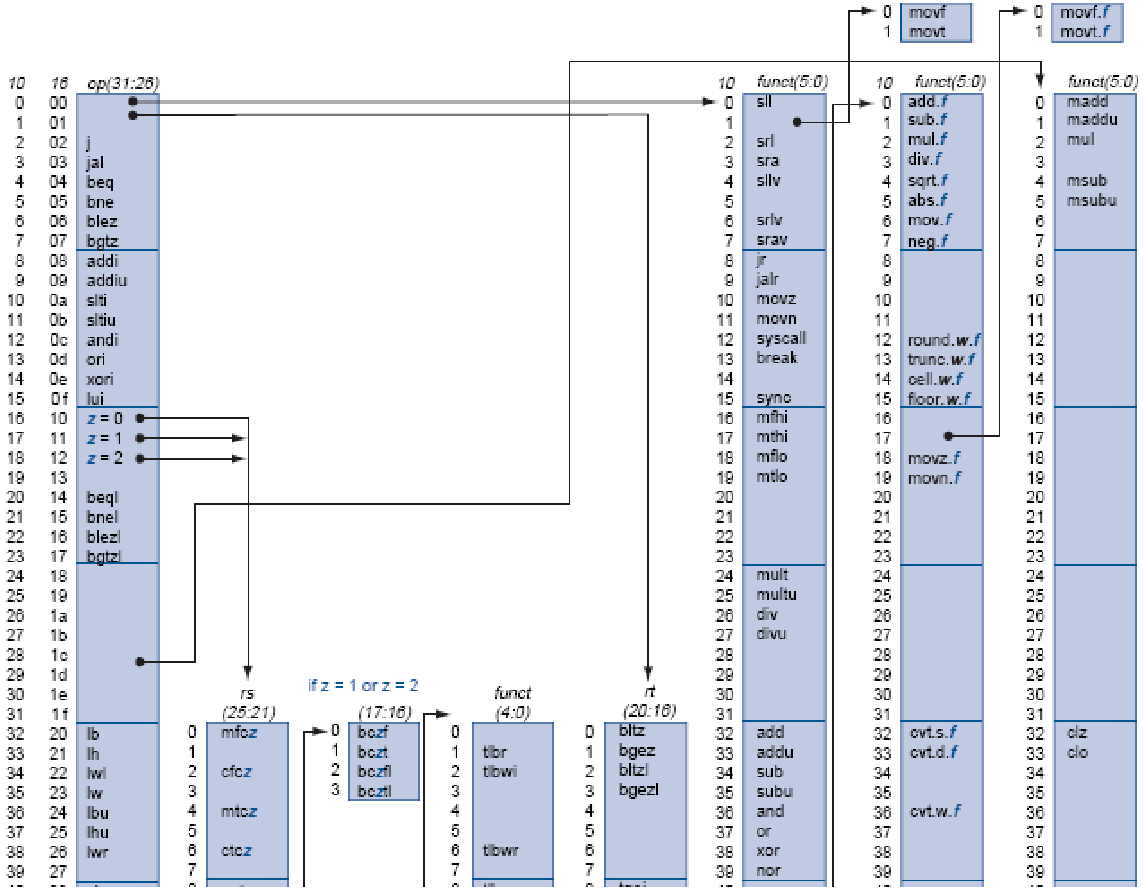
	(17:16)
0	bc*z*f
1	bc*z*t
2	bc*z*fl
3	bc*z*tl

FIGURE A.10.2 MIPS opcode map. The values of each field are shown to its left. The first column shows the values in base 10 and the second shows base 16 for the op field (bits 31 to 26) in the third column. This op field completely specifies the MIPS operation except for 6 op values: 0, 1, 16, 17, 18, and 19. These operations are determined by other fields, identified by pointers. The last field (funct) uses "*f*" to mean "s" if rs = 16 and op = 17 or "d" if rs = 17 and op = 17. The second field (rs) uses "z" to mean "0", "1", "2", or "3" if op = 16, 17, 18, or 19, respectively. If rs = 16, the operation is specified elsewhere: if z = 0, the operations are specified in the fourth field (bits 4 to 0); if z = 1, then the operations are in the last field with *f* = s. If rs = 17 and z = 1, then the operations are in the last field with *f* = d.

Machine Code Exercises (1)

- **Example: add $t0, $s1, $s2**
- **R-Format**

	op	rs	rt	rd	shamt	funct
R						
	6	5	5	5	5	6



10
16
op(31:26)
j
jal
beq
bne
blez
bgtz
addi
addiu
slti
sltiu
andi
ori
xori
lui
z = 0
z = 1
z = 2
beql
bnel
blezl
bgtzl
lb
lh
lwl
lw
lbu
lhu
lwr
rs
(25:21)
mfcz
cfcz
mtcz
ctcz
if z = 1 or z = 2
(17:16)
bczf
bczt
bczfl
bcztl
funct
(4:0)
tlbr
tlbwi
tlbwr
rt
(20:16)
bltz
bgez
bltzl
bgezl
funct(5:0)
sll
srl
sra
sllv
srlv
srav
jr
jalr
movz
movn
syscall
break
sync
mfhi
mthi
mflo
mtlo
mult
multu
div
divu
add
addu
sub
subu
and
or
xor
nor
movf
movt
funct(5:0)
add.f
sub.f
mul.f
div.f
sqrt.f
abs.f
mov.f
neg.f
round.w.f
trunc.w.f
ceil.w.f
floor.w.f
movz.f
movn.f
cvt.s.f
cvt.d.f
cvt.w.f
movf.f
movt.f
funct(5:0)
madd
maddu
mul
msub
msubu
clz
clo

$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Code Exercises (1) -Ans.

8 17 18

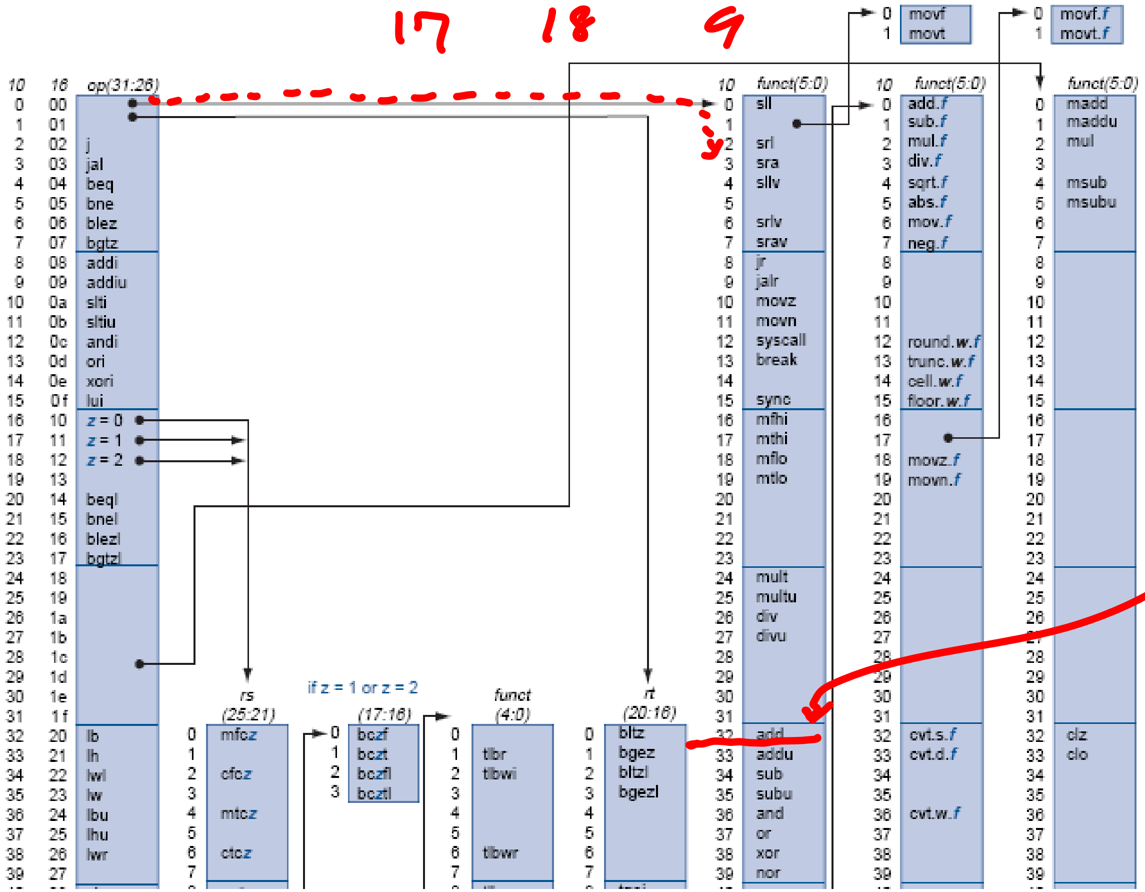
- Example: add $t0, $s1, $s2

dst src1 src2 (rt)

- R-Format

(32)

	op	rs	rt	rd	shamt	funct
R	00000	10001	10010	01000	00000	100000
	6	5	5	5	5	6
		17	18	9		

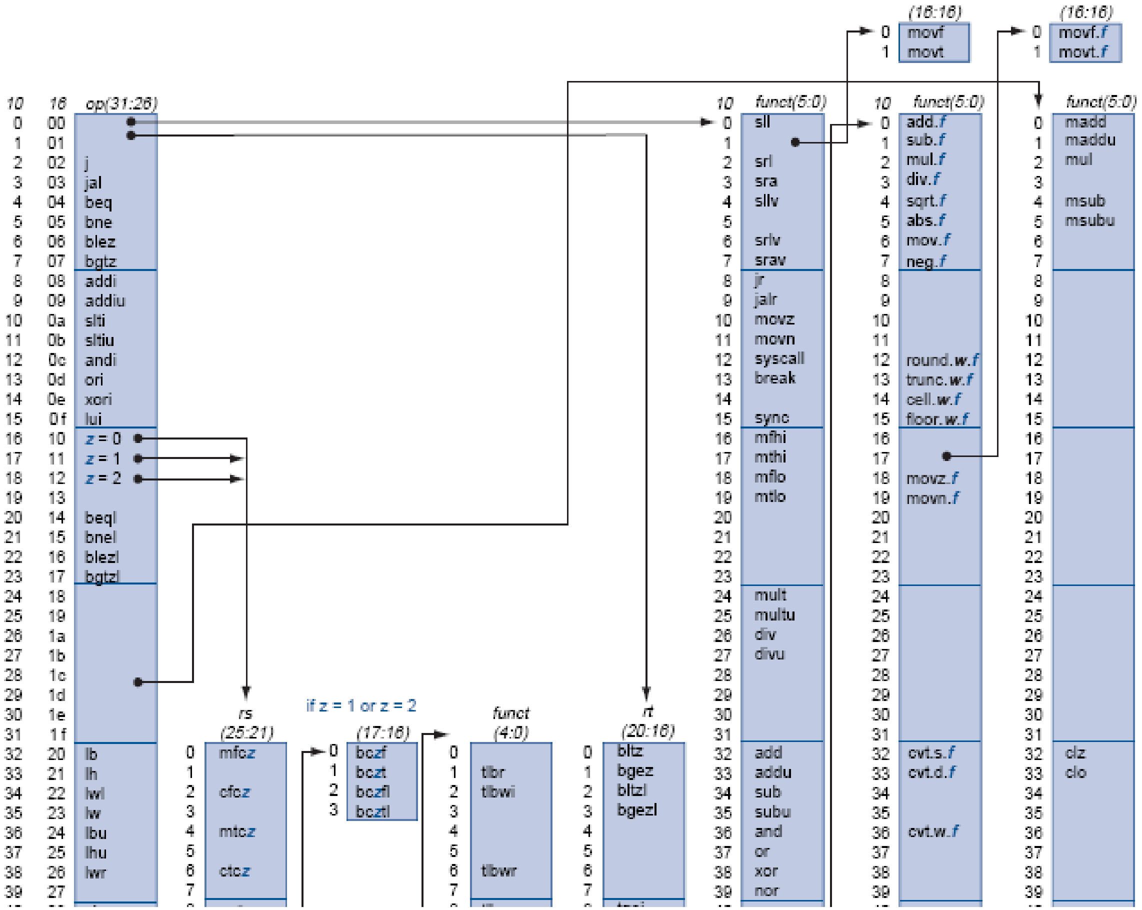


$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Code Exercises (2)

- **Example: add $s1, $s2, $s3**

	op	rs	rt	rd	shamt	funct
R						
	6	5	5	5	5	6



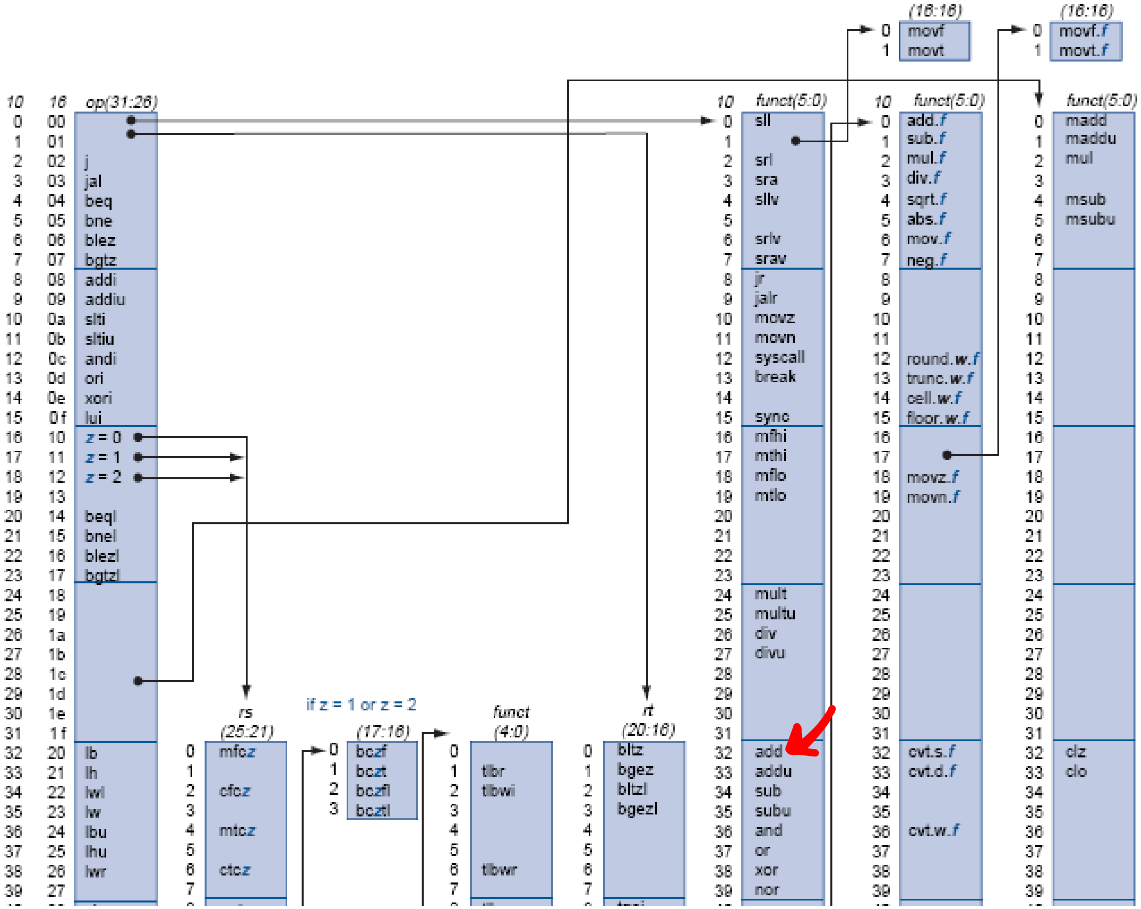
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Code Exercises (2)-Ans.

rd rs rt

- **Example: add $s1, $s2, $s3**

	op	rs	rt	rd	shamt	funct
R	0	18	19	17	0	32
	6	5	5	5	5	6



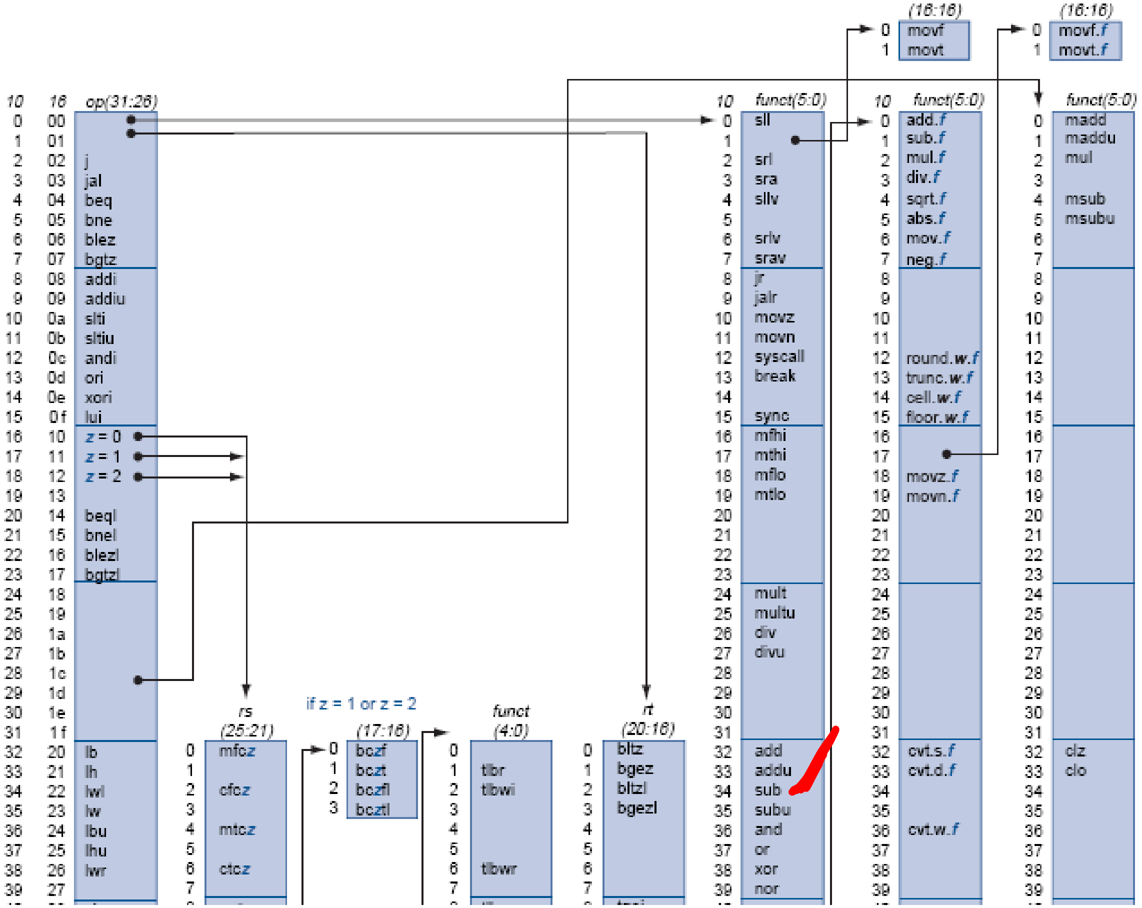
10 16 op(31:26)
0 00, 1 01, 2 02 j, 3 03 jal, 4 04 beq, 5 05 bne, 6 06 blez, 7 07 bgtz
8 08 addi, 9 09 addiu, 10 0a slti, 11 0b sltiu, 12 0c andi, 13 0d ori, 14 0e xori, 15 0f lui
16 10 z = 0, 17 11 z = 1, 18 12 z = 2, 19 13
20 14 beql, 21 15 bnel, 22 16 blezl, 23 17 bgtzl
24 18, 25 19, 26 1a, 27 1b, 28 1c, 29 1d, 30 1e, 31 1f
32 20 lb, 33 21 lh, 34 22 lwl, 35 23 lw, 36 24 lbu, 37 25 lhu, 38 26 lwr, 39 27
rs (25:21): 0 mfcz, 2 cfcz, 4 mtcz, 6 ctcz
if z = 1 or z = 2
(17:16): 0 bczf, 1 bczt, 2 bczfl, 3 bcztl
funct (4:0): 1 tlbr, 2 tlbwi, 6 tlbwr
rt (20:16): 0 bltz, 1 bgez, 2 bltzl, 3 bgezl
10 funct(5:0): 0 sll, 2 srl, 3 sra, 4 sllv, 6 srlv, 7 srav, 8 jr, 9 jalr, 10 movz, 11 movn, 12 syscall, 13 break, 15 sync, 16 mfhi, 17 mthi, 18 mflo, 19 mtlo, 24 mult, 25 multu, 26 div, 27 divu, 32 add, 33 addu, 34 sub, 35 subu, 36 and, 37 or, 38 xor, 39 nor
(16:16): 0 movf, 1 movt
10 funct(5:0): 0 add.f, 1 sub.f, 2 mul.f, 3 div.f, 4 sqrt.f, 5 abs.f, 6 mov.f, 7 neg.f, 12 round.w.f, 13 trunc.w.f, 14 cell.w.f, 15 floor.w.f, 18 movz.f, 19 movn.f, 32 cvt.s.f, 33 cvt.d.f, 36 cvt.w.f
(16:16): 0 movf.f, 1 movt.f
funct(5:0): 0 madd, 1 maddu, 2 mul, 4 msub, 5 msubu, 32 clz, 33 clo

$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Code Exercises (3)

- **Example: sub $s1, $s2, $s3**

	op	rs	rt	rd	shamt	funct
R						
	6	5	5	5	5	6

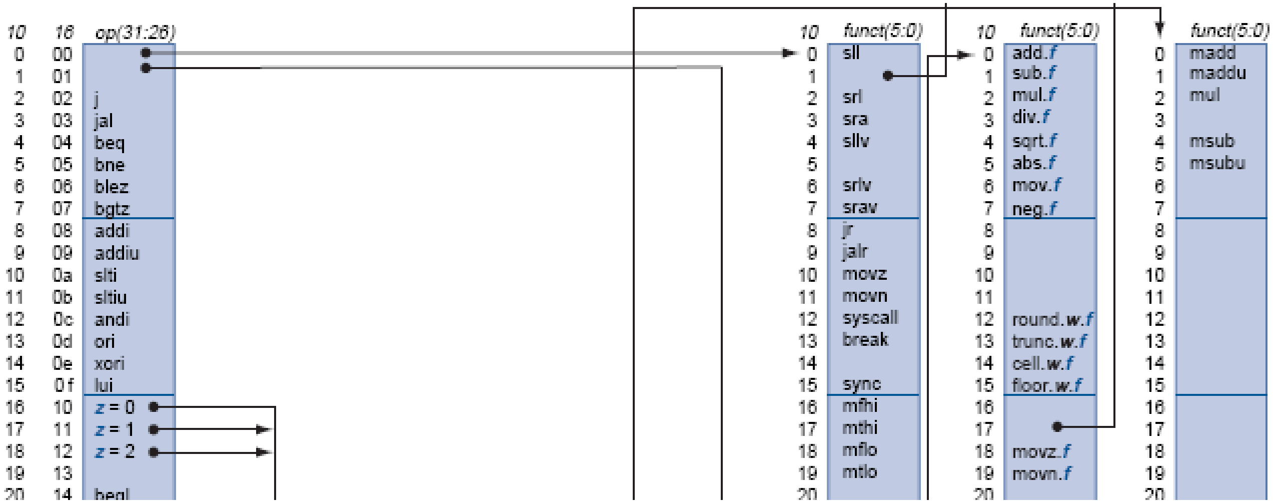


$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Code Exercises (4)

- **Example: addi $s1, $s2, 100**

	op	rs	rt	Offset
I				
	6	5	5	16



10	16	op(31:26)
0	00	
1	01	
2	02	j
3	03	jal
4	04	beq
5	05	bne
6	06	blez
7	07	bgtz
8	08	addi
9	09	addiu
10	0a	slti
11	0b	sltiu
12	0c	andi
13	0d	ori
14	0e	xori
15	0f	lui
16	10	*z* = 0
17	11	*z* = 1
18	12	*z* = 2
19	13	
20	14	beql

10	funct(5:0)
0	sll
1	
2	srl
3	sra
4	sllv
5	
6	srlv
7	srav
8	jr
9	jalr
10	movz
11	movn
12	syscall
13	break
14	
15	sync
16	mfhi
17	mthi
18	mflo
19	mtlo
20	

10	funct(5:0)
0	add.*f*
1	sub.*f*
2	mul.*f*
3	div.*f*
4	sqrt.*f*
5	abs.*f*
6	mov.*f*
7	neg.*f*
8	
9	
10	
11	
12	round.w.*f*
13	trunc.w.*f*
14	cell.w.*f*
15	floor.w.*f*
16	
17	
18	movz.*f*
19	movn.*f*
20	

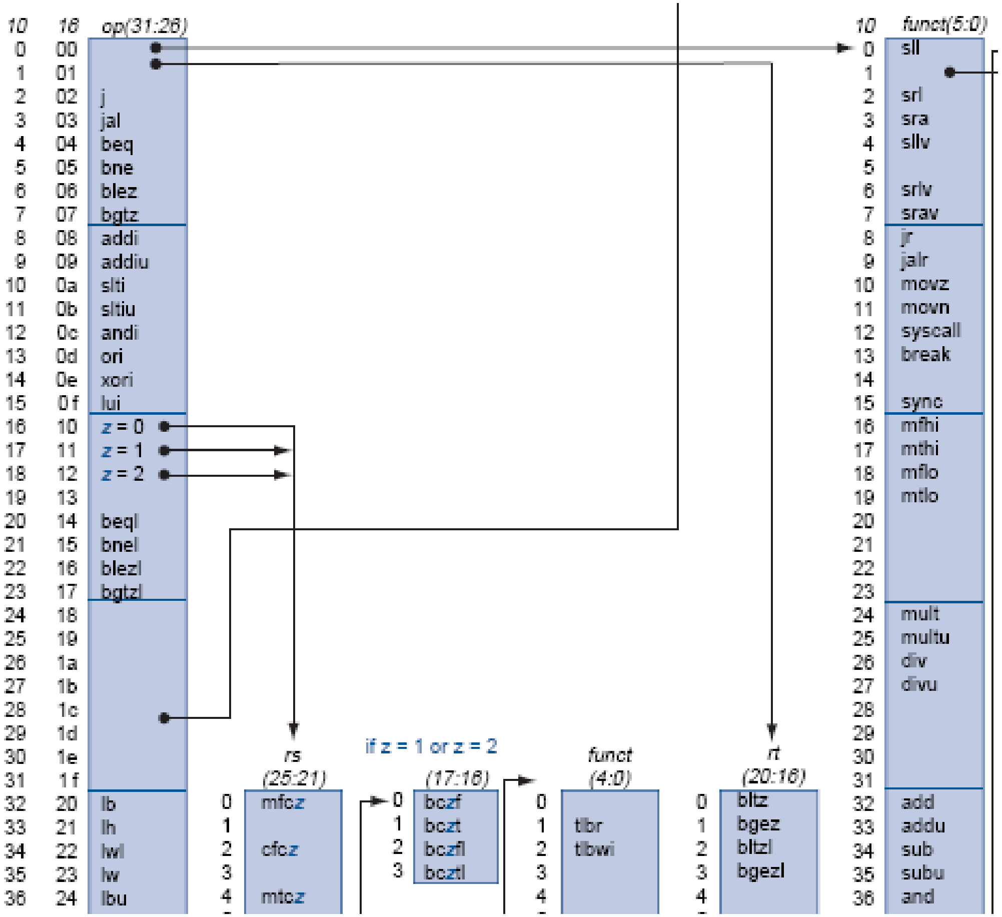
	funct(5:0)
0	madd
1	maddu
2	mul
3	
4	msub
5	msubu
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	

$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Code Exercises (5)

- **Example: lw $s1, 100($s2)**

	op	rs	rt	Offset
I				
	6	5	5	16



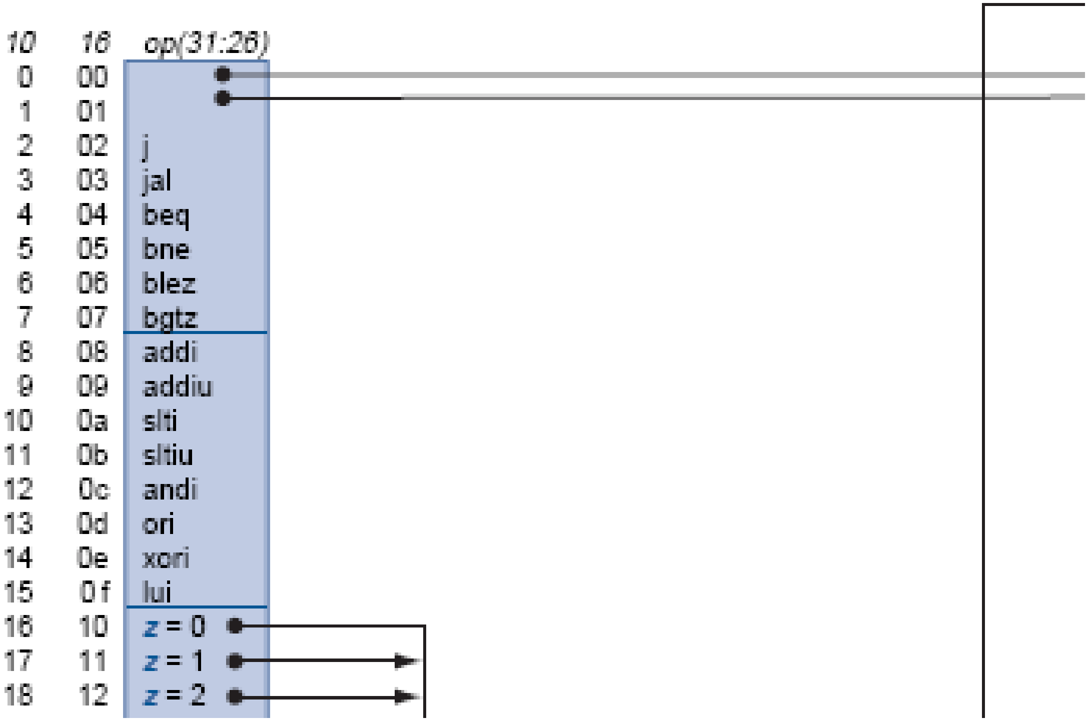
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Code Exercises (6)

Andi rt, rs, 'Imm'

- **Example: andi $s1, $s2, 100**

	op	rs	rt	Offset
I				
	6	5	5	16



10	16	op(31:26)
0	00	
1	01	
2	02	j
3	03	jal
4	04	beq
5	05	bne
6	06	blez
7	07	bgtz
8	08	addi
9	09	addiu
10	0a	slti
11	0b	sltiu
12	0c	andi
13	0d	ori
14	0e	xori
15	0f	lui
16	10	*z* = 0
17	11	*z* = 1
18	12	*z* = 2

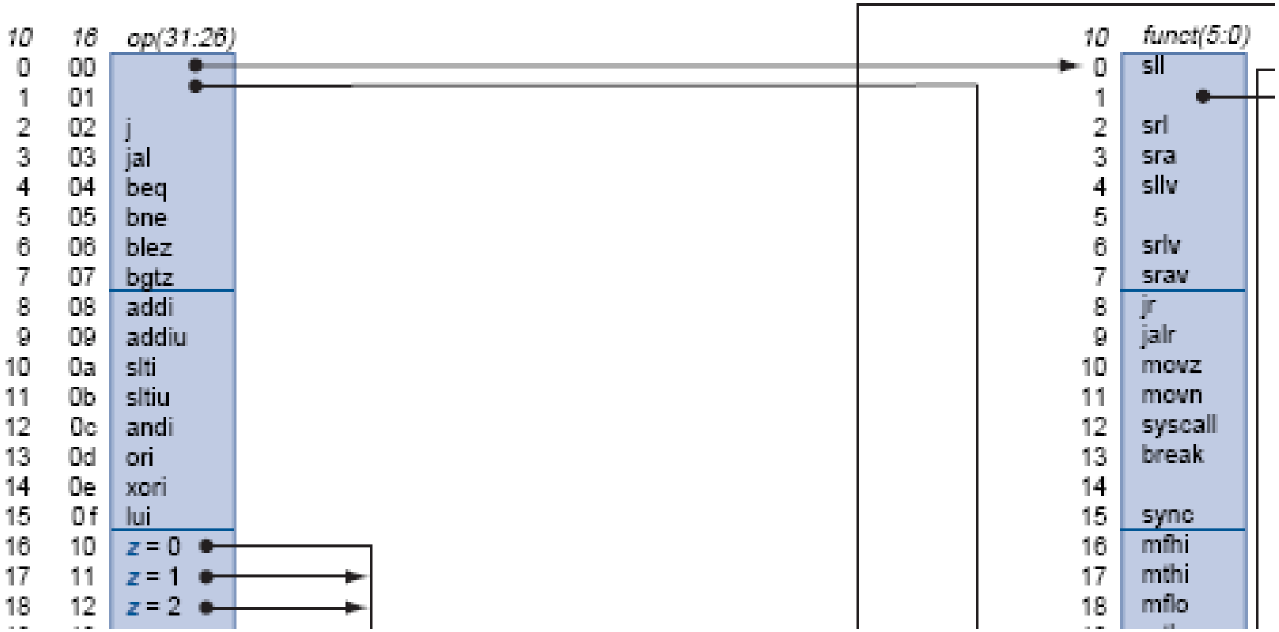
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Code Exercises (7)

Note* $\boxed{\text{sll rd, rt, shamt}}$ no rs

- **Example: sll $s1, $s2, 10**

	op	rs	rt	rd	shamt	funct
R						
	6	5	5	5	5	6



10	16	op(31:26)
0	00	
1	01	
2	02	j
3	03	jal
4	04	beq
5	05	bne
6	06	blez
7	07	bgtz
8	08	addi
9	09	addiu
10	0a	slti
11	0b	sltiu
12	0c	andi
13	0d	ori
14	0e	xori
15	0f	lui
16	10	*z* = 0
17	11	*z* = 1
18	12	*z* = 2

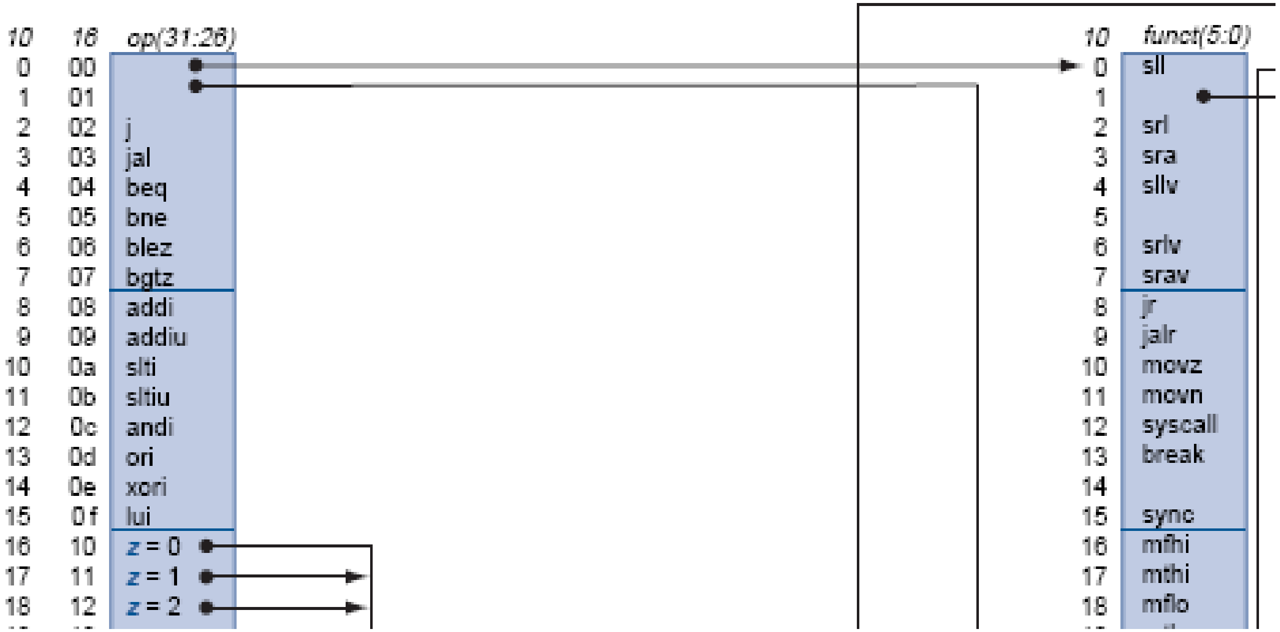
10	funct(5:0)
0	sll
1	
2	srl
3	sra
4	sllv
5	
6	srlv
7	srav
8	jr
9	jalr
10	movz
11	movn
12	syscall
13	break
14	
15	sync
16	mfhi
17	mthi
18	mflo

$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Code Exercises (8)

- **Example: beq $s1, $s2, 100**

	op	rs	rt	Offset
I				
	6	5	5	16



10	16	op(31:26)
0	00	
1	01	
2	02	j
3	03	jal
4	04	beq
5	05	bne
6	06	blez
7	07	bgtz
8	08	addi
9	09	addiu
10	0a	slti
11	0b	sltiu
12	0c	andi
13	0d	ori
14	0e	xori
15	0f	lui
16	10	z = 0
17	11	z = 1
18	12	z = 2

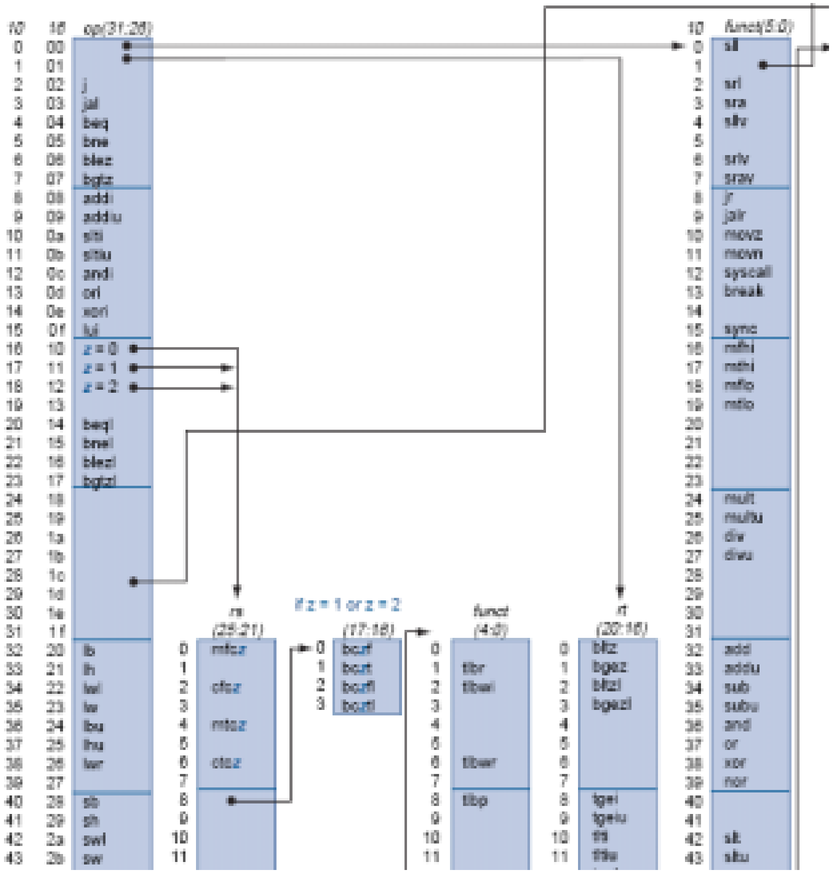
10	funct(5:0)
0	sll
1	
2	srl
3	sra
4	sllv
5	
6	srlv
7	srav
8	jr
9	jalr
10	movz
11	movn
12	syscall
13	break
14	
15	sync
16	mfhi
17	mthi
18	mflo

$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Code Exercises (9)

- **Example: slt $s1, $s2, $s3**

	op	rs	rt	rd	shamt	funct
R						
	6	5	5	5	5	6

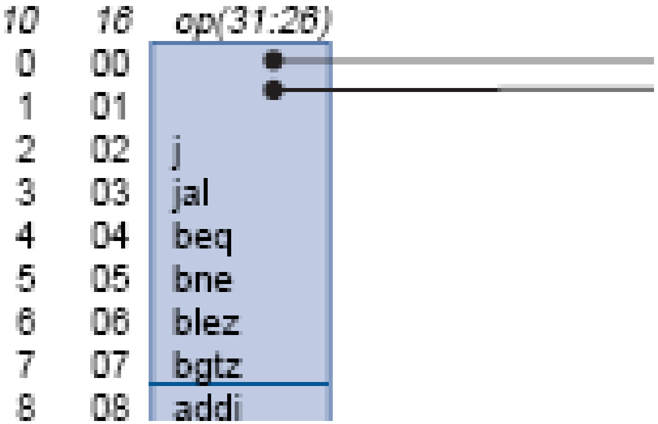


$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Machine Code Exercises (10)

- **Example: j 10000**

	op	Target
J		
	6	26



10	16	op(31:26)
0	00	
1	01	
2	02	j
3	03	jal
4	04	beq
5	05	bne
6	06	blez
7	07	bgtz
8	08	addi

$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29
$fp	30
$ra	31

Instruction and Machine Language - summary

-

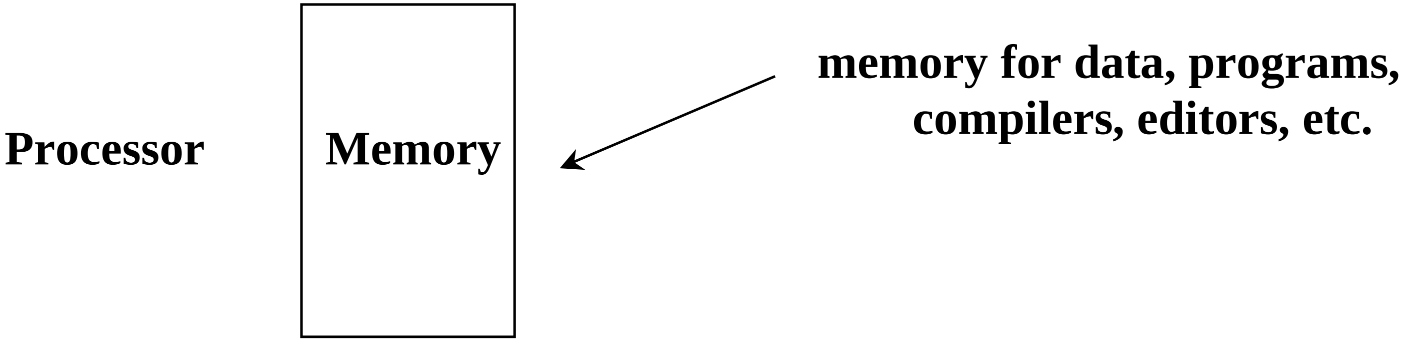
Instruction	**Meaning**
`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`
`sub $s1,$s2,$s3`	`$s1 = $s2 – $s3`
`lw $s1,100($s2)`	`$s1 = Memory[$s2+100]`
`sw $s1,100($s2)`	`Memory[$s2+100] = $s1`
`bne $s4,$s5,L`	`Next instr. is at Label if $s4 ≠ $s5`
`beq $s4,$s5,L`	`Next instr. is at Label if $s4 = $s5`
`j Label`	`Next instr. is at Label`

- Formats:

R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	16 bit address		
J	op	26 bit address				

Stored Program Concept

- Instructions are bits
- Programs are stored in memory
  — to be read or written just like data



- Fetch & Execute Cycle
  - Instructions are fetched and put into a special register
  - Bits in the register "control" the subsequent actions
  - Fetch the “next” instruction and continue

Control

- Decision making instructions
  - alter the control flow,
  - i.e., change the "next" instruction to be executed

- MIPS conditional branch instructions:

```
bne $t0, $t1, Label
beq $t0, $t1, Label
```

- Example: `if (i==j) h = i + j;`

```
          bne $s0, $s1, Label
          add $s3, $s0, $s1
   Label: ....
```

Control

- **MIPS unconditional branch instructions:**

```
j   label
```

- **Example:**

```
if (i!=j)               beq $s4, $s5, Lab1
    h=i+j;              add $s3, $s4, $s5
else                    j Lab2
    h=i-j;              Lab1:  sub $s3, $s4, $s5
                        Lab2:  ...
```

- *Can you build a simple for loop?*

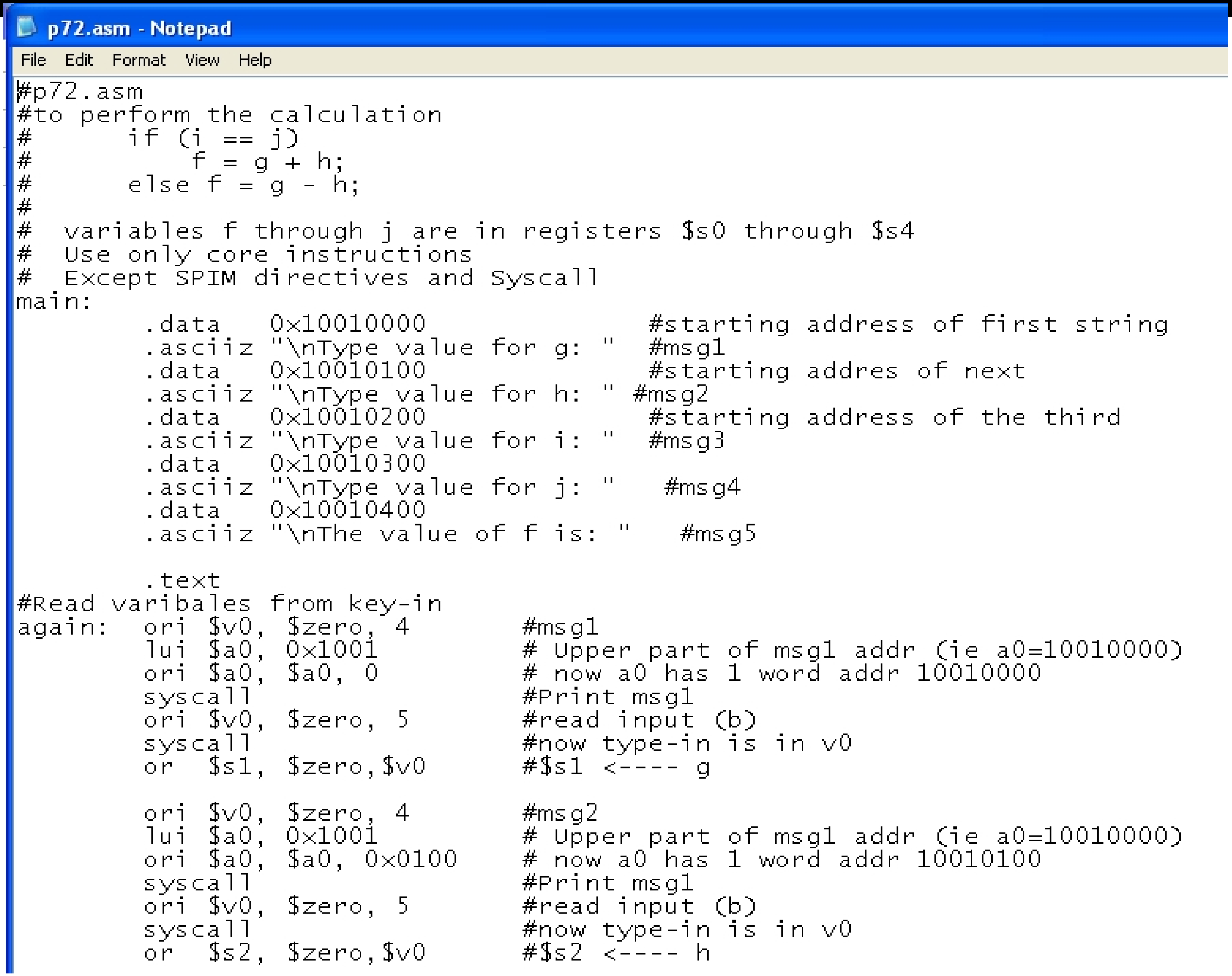
Control Flow

- We have: beq, bne, what about Branch-if-less-than?
- New instruction:

```
                                      if  $s1 < $s2 then
                                         $t0 = 1
     slt $t0, $s1, $s2                else
                                         $t0 = 0
```

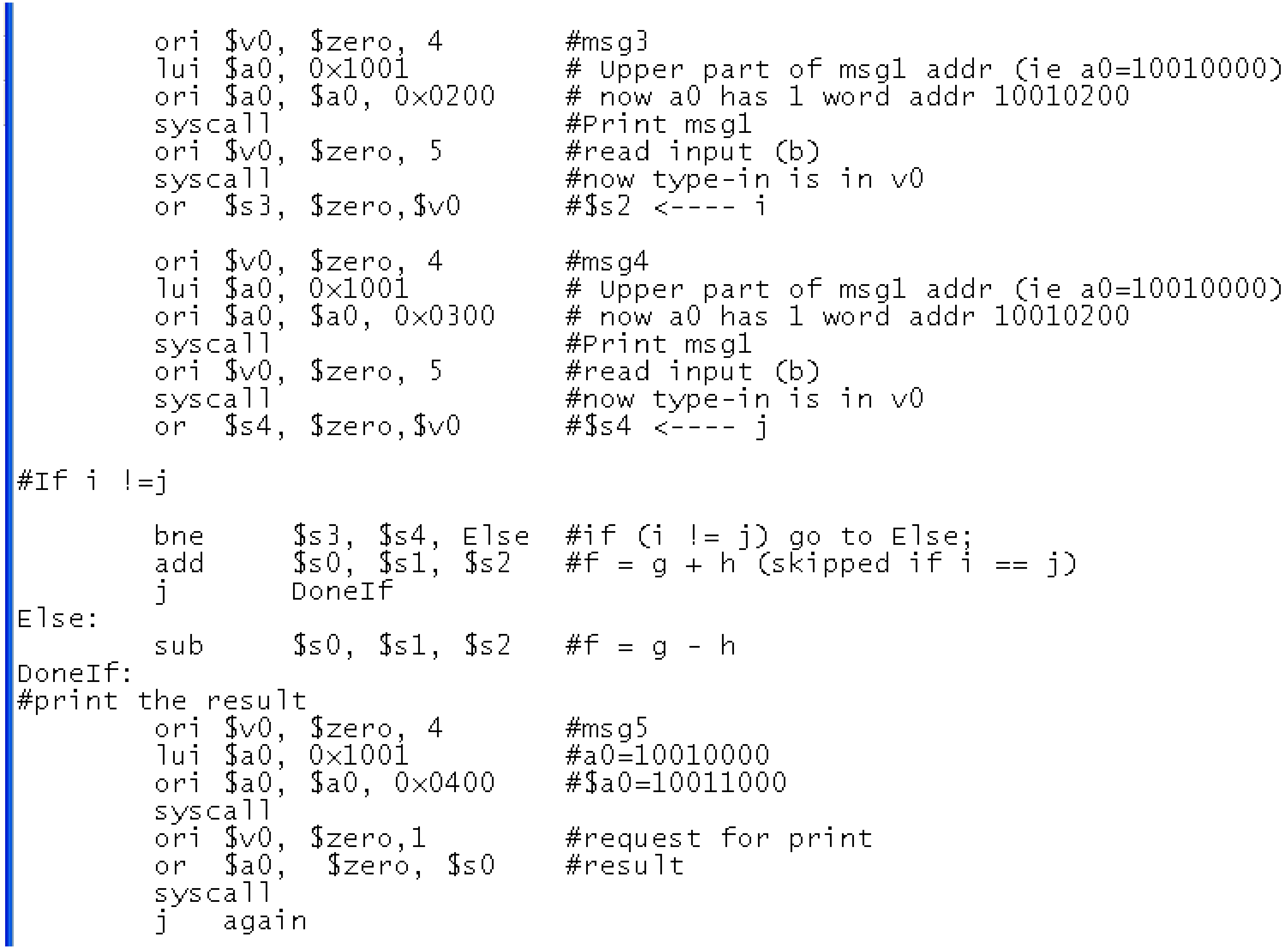
- Can use this instruction to build `"blt $s1, $s2, Label"`
  — can now build general control structures
- Note that the assembler needs a register to do this,
  — there are policy of use conventions for registers

Decision Making Exercise (p.72) 1/2



```
#p72.asm
#to perform the calculation
#      if (i == j)
#          f = g + h;
#      else f = g - h;
#
#  variables f through j are in registers $s0 through $s4
#  Use only core instructions
#  Except SPIM directives and Syscall
main:
        .data   0x10010000              #starting address of first string
        .asciiz "\nType value for g: "   #msg1
        .data   0x10010100              #starting addres of next
        .asciiz "\nType value for h: "  #msg2
        .data   0x10010200              #starting address of the third
        .asciiz "\nType value for i: "   #msg3
        .data   0x10010300
        .asciiz "\nType value for j: "    #msg4
        .data   0x10010400
        .asciiz "\nThe value of f is: "   #msg5

        .text
#Read varibales from key-in
again:  ori $v0, $zero, 4       #msg1
        lui $a0, 0x1001         # Upper part of msg1 addr (ie a0=10010000)
        ori $a0, $a0, 0         # now a0 has 1 word addr 10010000
        syscall                 #Print msg1
        ori $v0, $zero, 5       #read input (b)
        syscall                 #now type-in is in v0
        or  $s1, $zero,$v0      #$s1 <---- g

        ori $v0, $zero, 4       #msg2
        lui $a0, 0x1001         # Upper part of msg1 addr (ie a0=10010000)
        ori $a0, $a0, 0x0100    # now a0 has 1 word addr 10010100
        syscall                 #Print msg1
        ori $v0, $zero, 5       #read input (b)
        syscall                 #now type-in is in v0
        or  $s2, $zero,$v0      #$s2 <---- h
```

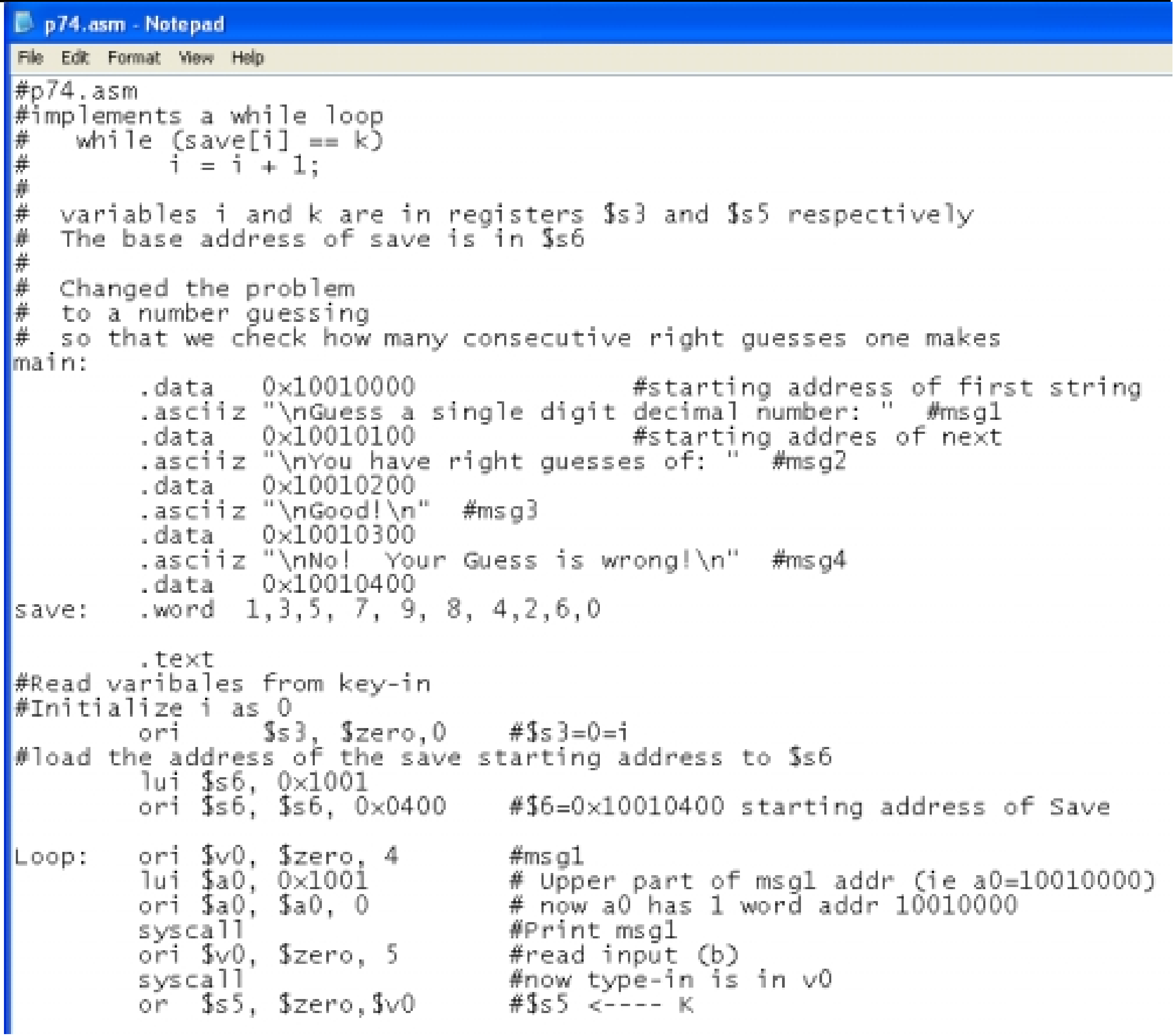
Decision Making Exercise (p.72) 2/2



```
        ori $v0, $zero, 4       #msg3
        lui $a0, 0x1001         # Upper part of msg1 addr (ie a0=10010000)
        ori $a0, $a0, 0x0200    # now a0 has 1 word addr 10010200
        syscall                 #Print msg1
        ori $v0, $zero, 5       #read input (b)
        syscall                 #now type-in is in v0
        or  $s3, $zero,$v0      #$s2 <---- i

        ori $v0, $zero, 4       #msg4
        lui $a0, 0x1001         # Upper part of msg1 addr (ie a0=10010000)
        ori $a0, $a0, 0x0300    # now a0 has 1 word addr 10010200
        syscall                 #Print msg1
        ori $v0, $zero, 5       #read input (b)
        syscall                 #now type-in is in v0
        or  $s4, $zero,$v0      #$s4 <---- j

#If i !=j

        bne     $s3, $s4, Else  #if (i != j) go to Else;
        add     $s0, $s1, $s2   #f = g + h (skipped if i == j)
        j       DoneIf
Else:
        sub     $s0, $s1, $s2   #f = g - h
DoneIf:
#print the result
        ori $v0, $zero, 4       #msg5
        lui $a0, 0x1001         #a0=10010000
        ori $a0, $a0, 0x0400    #$a0=10011000
        syscall
        ori $v0, $zero,1        #request for print
        or  $a0,  $zero, $s0    #result
        syscall
        j   again
```

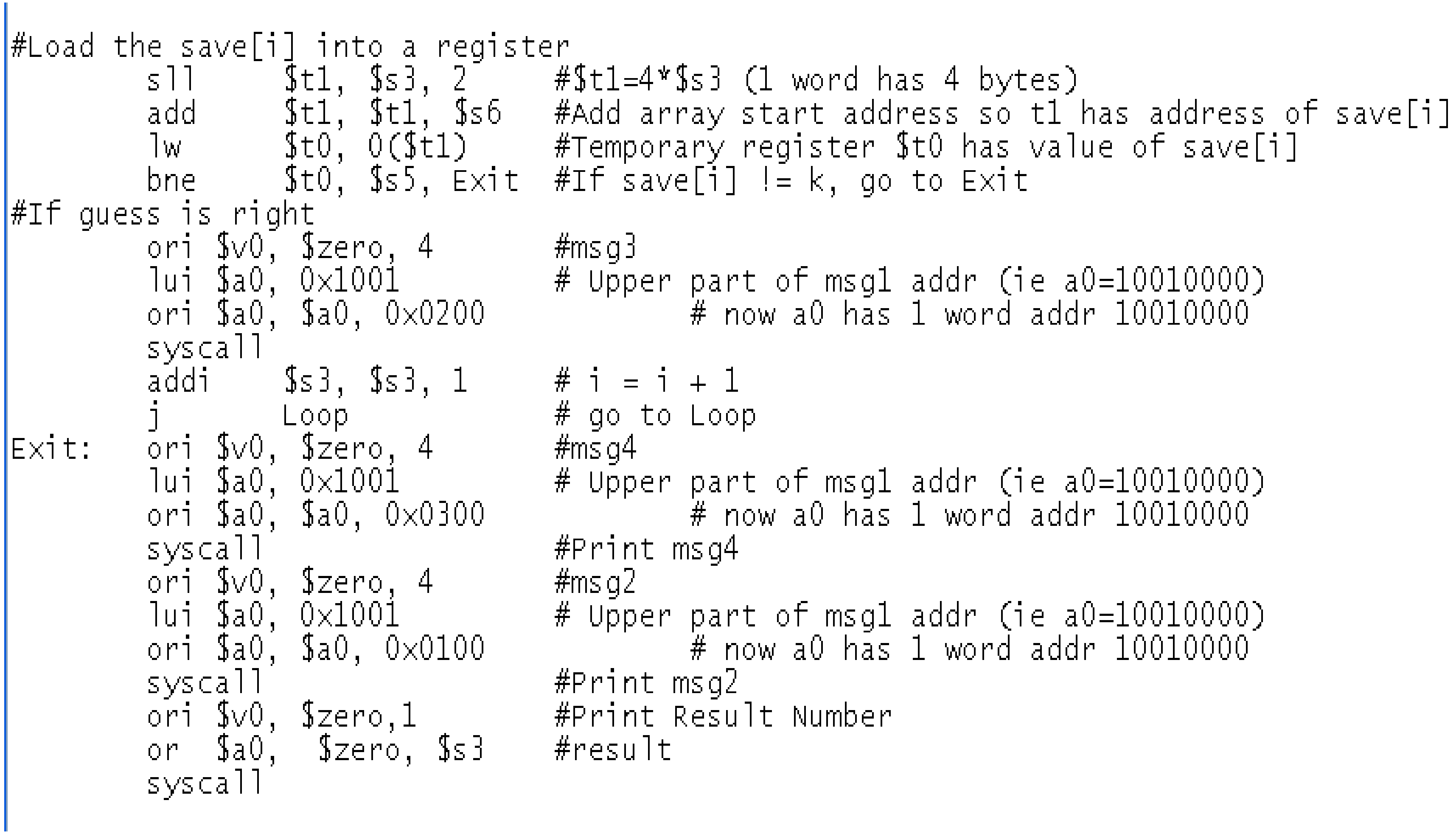
Decision Making Instruction (p74.asm) p1/2

- Ten single digit decimal number are stored at Save[i]
- Guess a number



```
#p74.asm
#implements a while loop
#   while (save[i] == k)
#         i = i + 1;
#
#  variables i and k are in registers $s3 and $s5 respectively
#  The base address of save is in $s6
#
#  Changed the problem
#  to a number guessing
#  so that we check how many consecutive right guesses one makes
main:
        .data   0x10010000                 #starting address of first string
        .asciiz "\nGuess a single digit decimal number: "  #msg1
        .data   0x10010100                 #starting addres of next
        .asciiz "\nYou have right guesses of: "  #msg2
        .data   0x10010200
        .asciiz "\nGood!\n"  #msg3
        .data   0x10010300
        .asciiz "\nNo!  Your Guess is wrong!\n"  #msg4
        .data   0x10010400
save:   .word  1,3,5, 7, 9, 8, 4,2,6,0

        .text
#Read varibales from key-in
#Initialize i as 0
        ori     $s3, $zero,0    #$s3=0=i
#load the address of the save starting address to $s6
        lui $s6, 0x1001
        ori $s6, $s6, 0x0400    #$6=0x10010400 starting address of Save

Loop:   ori $v0, $zero, 4       #msg1
        lui $a0, 0x1001         # Upper part of msg1 addr (ie a0=10010000)
        ori $a0, $a0, 0         # now a0 has 1 word addr 10010000
        syscall                 #Print msg1
        ori $v0, $zero, 5       #read input (b)
        syscall                 #now type-in is in v0
        or  $s5, $zero,$v0      #$s5 <---- K
```

Decision Making Instruction (p74.asm) p2/2



```
#Load the save[i] into a register
        sll     $t1, $s3, 2     #$t1=4*$s3 (1 word has 4 bytes)
        add     $t1, $t1, $s6   #Add array start address so t1 has address of save[i]
        lw      $t0, 0($t1)     #Temporary register $t0 has value of save[i]
        bne     $t0, $s5, Exit  #If save[i] != k, go to Exit
#If guess is right
        ori $v0, $zero, 4       #msg3
        lui $a0, 0x1001         # Upper part of msg1 addr (ie a0=10010000)
        ori $a0, $a0, 0x0200            # now a0 has 1 word addr 10010000
        syscall
        addi    $s3, $s3, 1     # i = i + 1
        j       Loop            # go to Loop
Exit:   ori $v0, $zero, 4       #msg4
        lui $a0, 0x1001         # Upper part of msg1 addr (ie a0=10010000)
        ori $a0, $a0, 0x0300            # now a0 has 1 word addr 10010000
        syscall                 #Print msg4
        ori $v0, $zero, 4       #msg2
        lui $a0, 0x1001         # Upper part of msg1 addr (ie a0=10010000)
        ori $a0, $a0, 0x0100            # now a0 has 1 word addr 10010000
        syscall                 #Print msg2
        ori $v0, $zero,1        #Print Result Number
        or  $a0,  $zero, $s3    #result
        syscall
```

Constants

- Small constants are used quite frequently (50% of operands)
  e.g., A = A + 5;
  B = B + 1;
  C = C - 18;
- Solutions? Why not?
  - put 'typical constants' in memory and load them.
  - create hard-wired registers (like $zero) for constants like one.
- MIPS Instructions:

```
addi $29, $29, 4
slti $8, $18, 10
andi $29, $29, 6
ori $29, $29, 4
```

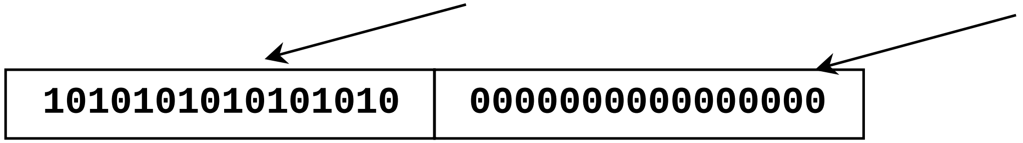
- Design Principle: Make the common case fast.

How about larger constants?

- We'd like to be able to load a 32 bit constant into a register
- Must use two instructions, new "load upper immediate" instruction

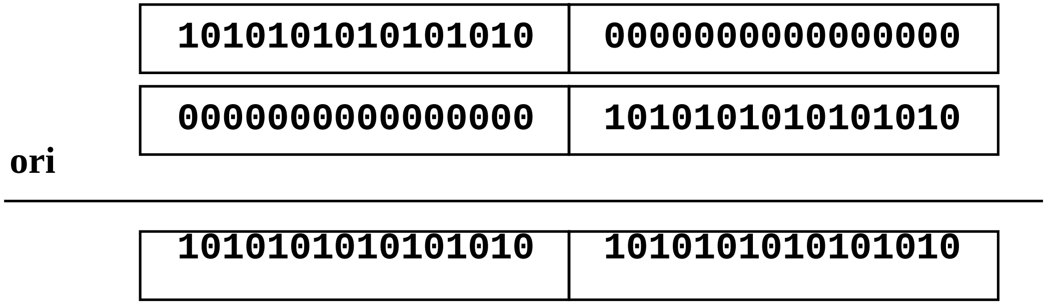
```
lui $t0, 1010101010101010
```

filled with zeros



- Then must get the lower order bits right, i.e.,

```
ori $t0, $t0, 1010101010101010
```



1010101010101010
0000000000000000
0000000000000000
1010101010101010
ori
1010101010101010
1010101010101010