Computer Architecture

Instruction Set Architecture

Branch Instructions

```
if (a == b)
    c = 1;

else
    c = 2;
```

Branch Instructions

```
if (a == b)
    c = 1;

else
    c = 2;
```

R5 = a
R6 = b
R7 = c

Branch Instructions

```
if (a == b)
    c = 1;

else
    c = 2;
```

R5 = a
R6 = b
R7 = c

```
    bne R5, R6, DoElse
        addi R7, R0, 1
        j SkipElse
DoElse:
        addi R7, R0, 2
SkipElse:
        …
```

Branch Instructions

```
if (a == b)
    c = 1;
else
    c = 2;
```

R5 = a
R6 = b
R7 = c

```
    beq R5, R6, Dolf
      addi R7, R0, 2
      j Skiplf
Dolf:
      addi R7, R0, 1
Skiplf:
      …
```

Loop

```
for (j = 1; j<=10; j++)
{
    c = c + j;
}
```

Loop

```
for (j = 1; j<=10; j++)
{
    c = c + j;
}
```

R2 = j

R3 = c

Loop

```
for (j = 1; j<=10; j++)
{
    c = c + j;
}
```

R2 = j

R3 = c

```
        addi R2, R0, 1
Loop:
        beq R2, 11, Exit
        add R3, R3, R2
        addi R2, R2, 1
        j Loop
Exit:
        …
```

Loop

```
for (j = 1; j<=10; j++)
{
    c = c + j;
}
```

R2 = j

R3 = c

```
    A) addi R2, R0, 1
Loop:
    B) beq R2, 11, Exit
    C) add R3, R3, R2
    D) addi R2, R2, 1
       j Loop
Exit:
       ...
    E) None of the above
```

Which of the lines of code above is wrong?
Write the letter corresponding to the line of code.

Loop

```
for (j = 1; j<=10; j++)
{
    c = c + j;
}
```

R2 = j

R3 = c

```
    A) addi R2, R0, 1
Loop:
    B) beq R2, 11, Exit
    C) add R3, R3, R2
    D) addi R2, R2, 1
       j Loop
Exit:
       ...
    E) None of the above
```

Which of the lines of code above is wrong?
Write the letter corresponding to the line of code.

Loop

```
for (j = 1; j<=10; j++)
{
    c = c + j;
}
```

R2 = j

R3 = c

Can only compare registers.
Cannot compare constants.

```
    A) addi R2, R0, 1
Loop:
    B) beq R2, 11, Exit
    C) add R3, R3, R2
    D) addi R2, R2, 1
       j Loop
Exit:
       ...
    E) None of the above
```

Which of the lines of code above is/are wrong if there's/re any?
Write the letter/s corresponding to the line/s of code.

Loop

```
for (j = 1; j<=10; j++)
{
    c = c + j;
}
```

R2 = j

R3 = c

```
        addi R2, R0, 1
        addi R1, R0, 11
Loop:
        beq R2, R1, Exit
        add R3, R3, R2
        addi R2, R2, 1
        j Loop
Exit:
        …
```

Instruction Encoding

- Computers do not understand “add R1, R2, R3”
- Instructions are translated to machine language (1s and 0s)

Instruction Encoding

For example,

add R8, R17, R18 is translated into machine language as

00000010 00110010 01000000 00100000

32 bits in size, that’s one word

Instruction Encoding

- MIPS Instructions have logical fields
- Write the machine code equivalent of the following MIPS Instruction according to its logical arrangement.

Note: Machine code for the opcode ‘add’ is 00000. The values of the *shamt* & *funct* fields may be disregarded.

add R3, R20, R31

Instruction Encoding

- MIPS Instructions have logical fields
- Write the machine code equivalent of the following MIPS Instruction according to its logical arrangement.

Note: Machine code for the opcode ‘add’ is 000000. The values of the *shamt* & *funct* fields may be disregarded.

add R3, R20, R31

opcode	rs (src1)	rt (src2)	rd (dest)	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Instruction Encoding

- MIPS Instructions have logical fields
- Write the machine code equivalent of the following MIPS Instruction according to its logical arrangement.

Note: Machine code for the opcode ‘add’ is 000000.

The values of the *shamt* & *funct* fields may be disregarded.

add R3, R20, R31

opcode	rs (src1)	rt (src2)	rd (dest)	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
000000	**10100**	**11111**	**00011**	**X**	**X**

MIPS Instruction Formats

MIPS has 3 instruction formats:

R	operation	3 registers	no immediate
I	operation	2 registers	16-bit immediate
J	*jump*	0 registers	26-bit immediate

MIPS Instruction Formats

MIPS has 3 instruction formats:

R	operation	3 registers	no immediate
I	operation	2 registers	16-bit immediate
J	*jump*	0 registers	26-bit immediate

Trade-off immediate space and registers:

Name	**Bit Fields**						**Notes**
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	(32 bits total)
R-Format	op	rs	rt	rd	shmt	funct	Arithmetic, logic
I-format	op	rs	rt	**address/immediate (16)**			Load/store, branch, immediate
J-format	op	**target address (26)**					Jump

MIPS Instruction Formats

Name	Bit Fields						Notes
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	(32 bits total)
R-Format	op	rs	rt	rd	shmt	funct	Arithmetic, logic
I-format	op	rs	rt	**address/immediate (16)**			Load/store, branch, immediate
J-format	op	**target address (26)**					Jump

For the I-format, the range of values that can be represented in the immediate field is from -2^{n-1} to $2^{n-1}-1$ (-32, 768 to 32, 767).

MIPS Instruction Formats

Consider the following ‘addi’ instruction, what is the 16-bit binary value in the immediate field when this instruction is executed?

addi R3, R5, -100

MIPS Instruction Formats

Consider the following ‘addi’ instruction, what is the 16-bit binary value in the immediate field when this instruction is executed?

addi R3, R5, -100

First, we convert 100 to its binary equivalent.

MIPS Instruction Formats

Consider the following ‘addi’ instruction, what is the 16-bit binary value in the immediate field when this instruction is executed?

addi R3, R5, -100

First, we convert 100 to its binary equivalent.

$100_{10} = 110\ 0100_2$

MIPS Instruction Formats

Consider the following ‘addi’ instruction, what is the 16-bit binary value in the immediate field when this instruction is executed?

addi R3, R5, -100

First, we convert 100 to its binary equivalent.

$100_{10} = 110\ 0100_2$

Sign extend to 16 bits. For positive 100, we sign extend 0.

MIPS Instruction Formats

Consider the following ‘addi’ instruction, what is the 16-bit binary value in the immediate field when this instruction is executed?

addi R3, R5, -100

First, we convert 100 to its binary equivalent.

$100_{10} = 110\ 0100_2$

Sign extend to 16 bits. For positive 100, we sign extend 0.

$0000\ 0000\ 0110\ 0100_2$

MIPS Instruction Formats

Consider the following ‘addi’ instruction, what is the 16-bit binary value in the immediate field when this instruction is executed?

addi R3, R5, -100

Get 2’s complement of 100_{10} ($0000\ 0000\ 0110\ 0100_2$).

$1111\ 1111\ 1001\ 1100_2$

MIPS Instruction Formats

Consider the following ‘addi’ instruction, what is the 16-bit binary value in the immediate field when this instruction is executed?

addi R3, R5, -100

Get 2’s complement of 100_{10} ($0000\ 0000\ 0110\ 0100_2$).

$1111\ 1111\ 1001\ 1100_2$

Another way is to get the 2’s complement first of -100 then sign extend 1.

MIPS Instruction Formats

Consider the following instructions stored in the memory:

Address	Instruction
12	addi R3, R0, 0
16	addi R4, R0, 7
20	bne R3, R4, 2
24	add R5, R5, R3
28	j 5
32	add R5, R5, R4

Suppose the current value of PC is 20, which instruction will be executed next after the current instruction is executed?

MIPS Instruction Formats

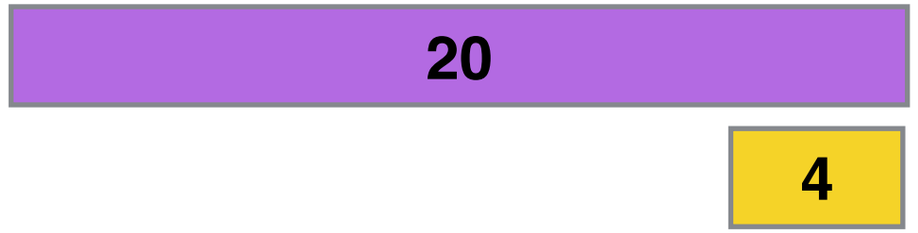
Consider the following instructions stored in the memory:

Address	Instruction
12	addi R3, R0, 0
16	addi R4, R0, 7
20	bne R3, R4, 2
24	add R5, R5, R3
28	j 5
32	add R5, R5, R4

MIPS Instruction Formats

Consider the following instructions stored in the memory:

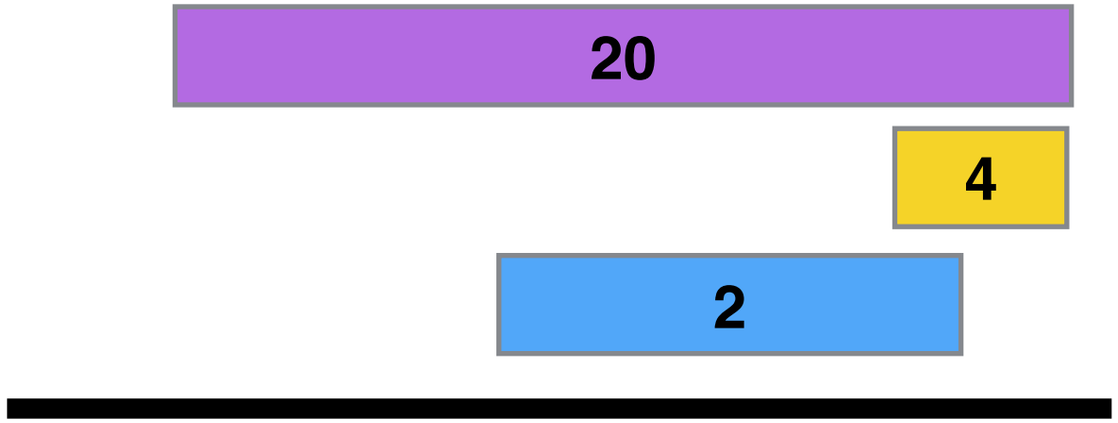
Address	Instruction
12	addi R3, R0, 0
16	addi R4, R0, 7
20	bne R3, R4, 2
24	add R5, R5, R3
28	j 5
32	add R5, R5, R4



MIPS Instruction Formats

Consider the following instructions stored in the memory:

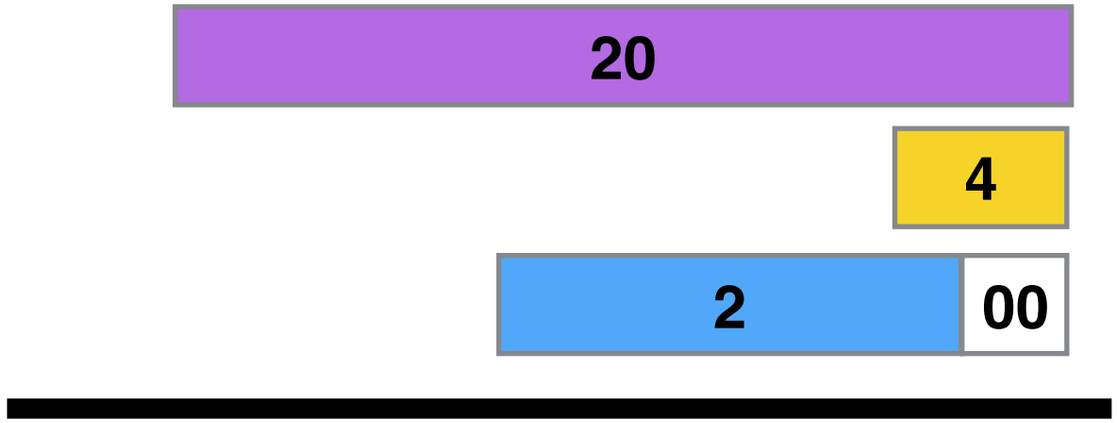
Address	Instruction
12	addi R3, R0, 0
16	addi R4, R0, 7
20	bne R3, R4, 2
24	add R5, R5, R3
28	j 5
32	add R5, R5, R4



MIPS Instruction Formats

Consider the following instructions stored in the memory:

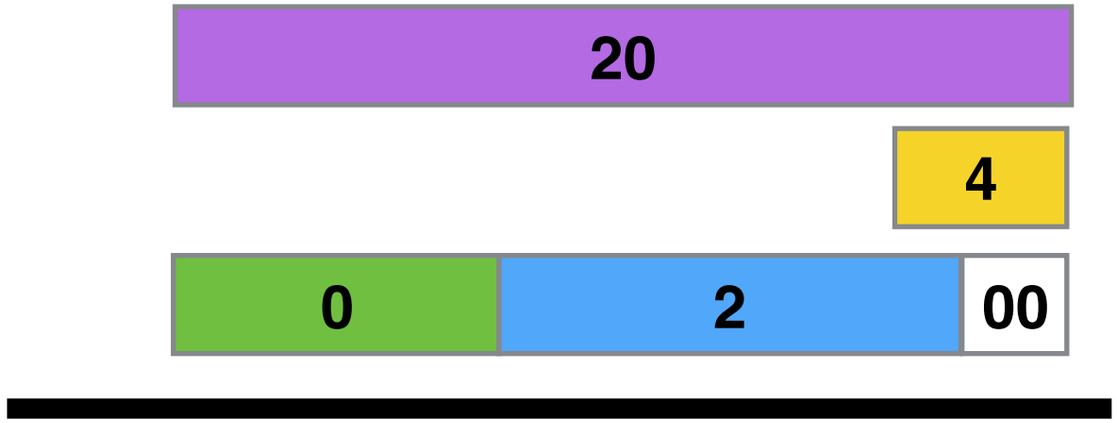
Address	Instruction
12	addi R3, R0, 0
16	addi R4, R0, 7
20	bne R3, R4, 2
24	add R5, R5, R3
28	j 5
32	add R5, R5, R4



MIPS Instruction Formats

Consider the following instructions stored in the memory:

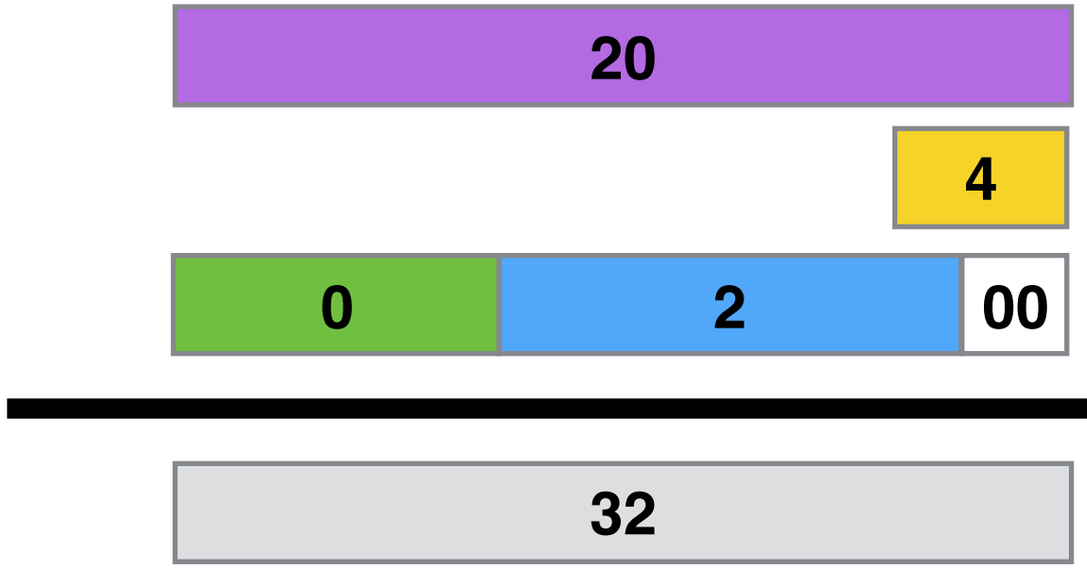
Address	Instruction
12	addi R3, R0, 0
16	addi R4, R0, 7
20	bne R3, R4, 2
24	add R5, R5, R3
28	j 5
32	add R5, R5, R4



MIPS Instruction Formats

Consider the following instructions stored in the memory:

Address	Instruction
12	addi R3, R0, 0
16	addi R4, R0, 7
20	bne R3, R4, 2
24	add R5, R5, R3
28	j 5
32	add R5, R5, R4



MIPS Instruction Formats

Consider the following instructions stored in the memory:

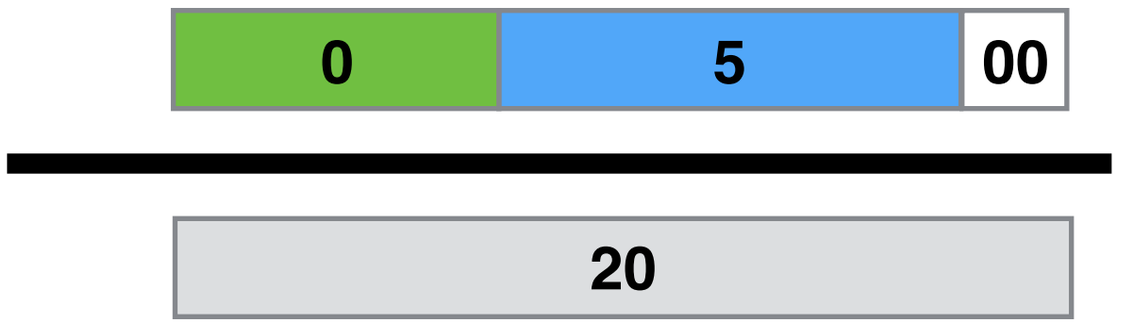
Address	Instruction
12	addi R3, R0, 0
16	addi R4, R0, 7
20	bne R3, R4, 2
24	add R5, R5, R3
28	j 5
32	add R5, R5, R4

Suppose the current value of PC is 28, which instruction will be executed next after the current instruction is executed?

MIPS Instruction Formats

Consider the following instructions stored in the memory:

Address	Instruction
12	addi R3, R0, 0
16	addi R4, R0, 7
20	bne R3, R4, 2
24	add R5, R5, R3
28	j 5
32	add R5, R5, R4



Large Constants

What is the content of R7 after the following MIPS instructions are executed?

ori R7, 00000000 11111111

lui R7, 11111111 00000000

Large Constants

What is the content of R7 after the following MIPS instructions are executed?

ori R7, 00000000 11111111 (load lower 16 bits)

lui R7, 11111111 00000000 (load upper 16 bits)

R7 (32 bits)	
upper 16 bits	*lower 16 bits*
11111111 00000000	**00000000 11111111**

Procedure

Procedures (functions/subroutines) are needed for structured programming

- to avoid repeated code
- to call functions you didn’t write (libraries)

Procedure

```
main()
{
    func1(arg1, arg2)
    …
}

func1(arg1, arg2)
{
    …
    return value
}
```

Procedure

```
main()
{
    func1(arg1, arg2)
    …
}

func1(arg1, arg2)
{
    …
    return value
}
```

we use labels to indicate procedures

we use the jal (jump-and-link) to call procedure

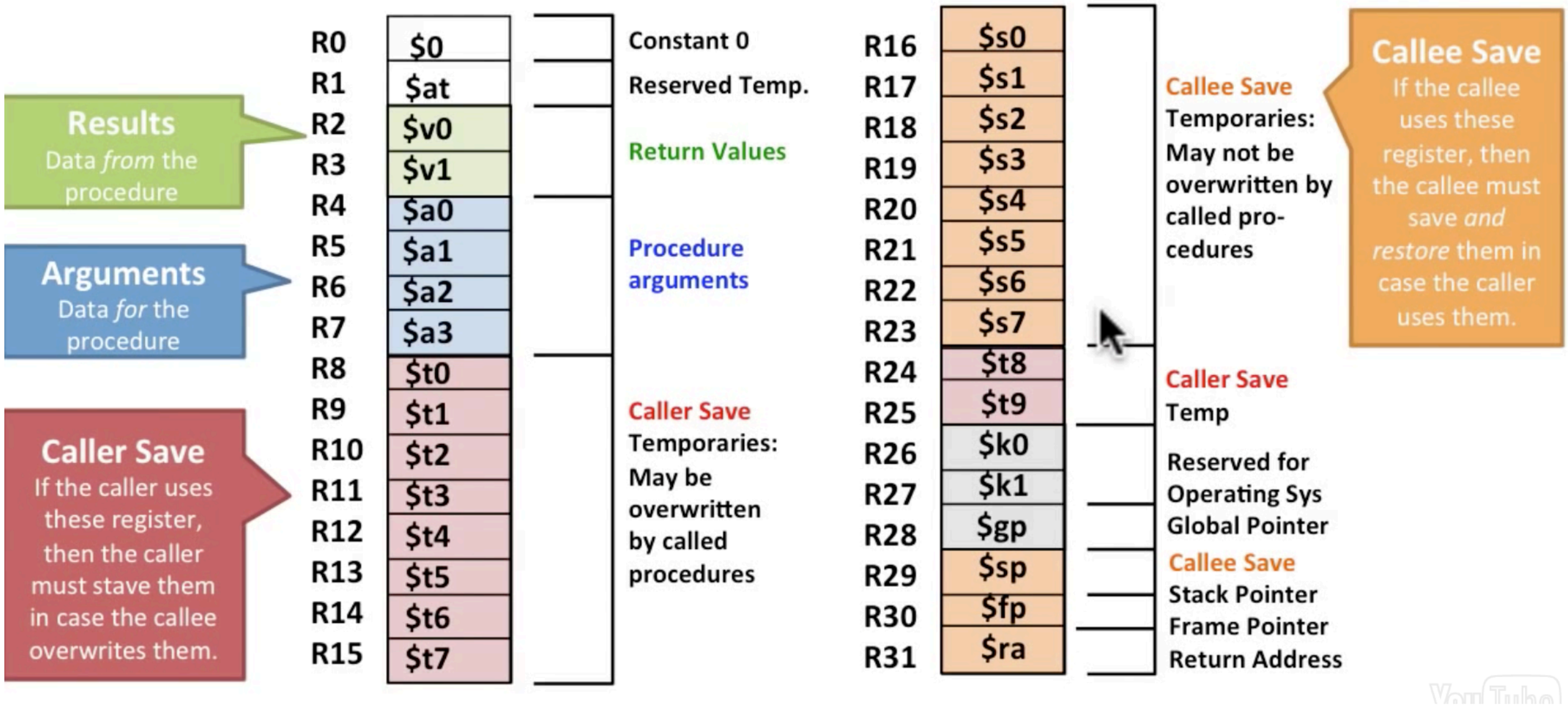
we use the jr (jump-return) to return to the caller

Procedure

```
main()
{
    func1(arg1, arg2)
    …
}

func1(arg1, arg2)
{
    …
    return
}
```

```
main:

    jal func1

func1:
    jr $ra
```

Procedure


Results
Data *from* the procedure
Arguments
Data *for* the procedure
Caller Save
If the caller uses these register, then the caller must stave them in case the callee overwrites them.
R0 $0
R1 $at
R2 $v0
R3 $v1
R4 $a0
R5 $a1
R6 $a2
R7 $a3
R8 $t0
R9 $t1
R10 $t2
R11 $t3
R12 $t4
R13 $t5
R14 $t6
R15 $t7
Constant 0
Reserved Temp.
Return Values
Procedure arguments
Caller Save
Temporaries:
May be overwritten by called procedures
R16 $s0
R17 $s1
R18 $s2
R19 $s3
R20 $s4
R21 $s5
R22 $s6
R23 $s7
R24 $t8
R25 $t9
R26 $k0
R27 $k1
R28 $gp
R29 $sp
R30 $fp
R31 $ra
Callee Save
Temporaries:
May not be overwritten by called pro-cedures
Caller Save
Temp
Reserved for Operating Sys
Global Pointer
Callee Save
Stack Pointer
Frame Pointer
Return Address
Callee Save
If the callee uses these register, then the callee must save *and restore* them in case the caller uses them.

Procedure

Consider the following MIPS code. What is the final value at R8 when the code is run?

```
main:
addi $a0, $0, 5
addi $a1, $0, 10

jal do_something

addi $t0, $v0, 10
```

```
do_something:
   add $s0, $a0, $0
add $s1, $a1, $0
add $s2, $0, $0
add $s3, $s2, $0

here:
   beq $s1, $s2, there
      add $s3, $s3, $s0
      addi $s2, $s2, 1
      j here

there:
   add $v0, $s3, $0
   jr $ra
```

Procedure

Consider the following MIPS code. What is the final value at R8 when the code is run?

```
main:
    addi $a0, $0, 5
    addi $a1, $0, 10

    jal do_something

    addi $t0, $v0, 10
```

```
do_something:
    add $s0, $a0, $0
    add $s1, $a1, $0
    add $s2, $0, $0
    add $s3, $s2, $0

here:
    beq $s1, $s2, there
        add $s3, $s3, $s0
        addi $s2, $s2, 1
        j here

there:
    add $v0, $s3, $0
    jr $ra
```

Procedure

Consider the following MIPS code. What is the final value at R8 when the code is run?

```
main:
addi $a0, $0, 5
addi $a1, $0, 10

jal do_something

addi $t0, $v0, 10
```

R8 = 50

```
do_something:
   add $s0, $a0, $0
   add $s1, $a1, $0
   add $s2, $0, $0
   add $s3, $s2, $0

here:
   beq $s1, $s2, there
      add $s3, $s3, $s0
      addi $s2, $s2, 1
      j here

there:
   add $v0, $s3, $0
   jr $ra
```

Procedure

Consider the following MIPS code. What is the final value of $t4 when the code is run?

```
main:
    addi $t0, $0, 3
    addi $t1, $0, 10
    addi $s2, $0, 8

    addi $sp, $sp, -8
    sw $t0, (0)$sp
    sw $t1, (4)$sp

    jal do_something

    lw $t0, (0)$sp
    lw $t1, (4)$sp
    addi $sp, $sp, 8

    add $t2, $s2, $t0
    add $s3, $v0, $t1
    add $t4, $t2, $s3
```

```
do_something:
    addi $t1, $0, 0
    addi $t0, $t1, 7
    add $s2, $t0, $t1
    add $v0, $t0, $s2

    jr $ra
```

Procedure

Consider the following MIPS code. What is the final value of $t4 when the code is run?

```
main:
    addi $t0, $0, 3
    addi $t1, $0, 10
    addi $s2, $0, 8

    addi $sp, $sp, -8
    sw $t0, (0)$sp
    sw $t1, (4)$sp

    jal do_something

    lw $t0, (0)$sp
    lw $t1, (4)$sp
    addi $sp, $sp, 8

    add $t2, $s2, $t0
    add $s3, $v0, $t1
    add $t4, $t2, $s3
```

```
do_something:
    addi $t1, $0, 0
    addi $t0, $t1, 7
    add $s2, $t0, $t1
    add $v0, $t0, $s2

    jr $ra
```

Procedure

Consider the following MIPS code. What is the final value of $t4 when the code is run?

```
main:
    addi $t0, $0, 3
    addi $t1, $0, 10
    addi $s2, $0, 8

    addi $sp, $sp, -8
    sw $t0, (0)$sp
    sw $t1, (4)$sp

    jal do_something

    lw $t0, (0)$sp
    lw $t1, (4)$sp
    addi $sp, $sp, 8

    add $t2, $s2, $t0
    add $s3, $v0, $t1
    add $t4, $t2, $s3
```

```
do_something:
    addi $t1, $0, 0
    addi $t0, $t1, 7
    add $s2, $t0, $t1
    add $v0, $t0, $s2

    jr $ra
```

$t4 = 34

ISAs

Choose the best answer.

	Machine Type		**Computer Type**	
	Register-Memory	*Load-Store*	*CISC*	*RISC*
A	x86	MIPS	MIPS	x86
B	MIPS	x86	MIPS	x86
C	x86	MIPS	x86	MIPS
D	MIPS	x86	x86	MIPS
E	None of the above			

ISAs

Choose the best answer.

	Machine Type		**Computer Type**	
	Register-Memory	*Load-Store*	*CISC*	*RISC*
A	x86	MIPS	MIPS	x86
B	MIPS	x86	MIPS	x86
C	x86	MIPS	x86	MIPS
D	MIPS	x86	x86	MIPS
E	None of the above			

Register-Memory: allows instructions to access both registers & memory

Load-store: instructions only allow registers

ISAs

Choose the best answer.

	Machine Type		Computer Type	
	Register-Memory	*Load-Store*	*CISC*	*RISC*
A	x86	MIPS	MIPS	x86
B	MIPS	x86	MIPS	x86
C	x86	MIPS	x86	MIPS
D	MIPS	x86	x86	MIPS
E	None of the above			

Register-Memory: allows instructions to access both registers & memory

Load-store: instructions only allow registers

ISAs

Choose the best answer.

	Machine Type		**Computer Type**	
	Register-Memory	*Load-Store*	*CISC*	*RISC*
A	x86	MIPS	MIPS	x86
B	MIPS	x86	MIPS	x86
C	x86	MIPS	x86	MIPS
D	MIPS	x86	x86	MIPS
E	None of the above			

CISC: Complex Instruction Set Computing

RISC: Reduced Instruction Set Computing

ISAs

Choose the best answer.

	Machine Type		**Computer Type**	
	Register-Memory	*Load-Store*	*CISC*	*RISC*
A	x86	MIPS	MIPS	x86
B	MIPS	x86	MIPS	x86
C	x86	MIPS	x86	MIPS
D	MIPS	x86	x86	MIPS
E	None of the above			

CISC: Complex Instruction Set Computing

RISC: Reduced Instruction Set Computing