Computer Architecture
Tutorial 2

Translation Hierarchy of a High-Level Program into Machine Code

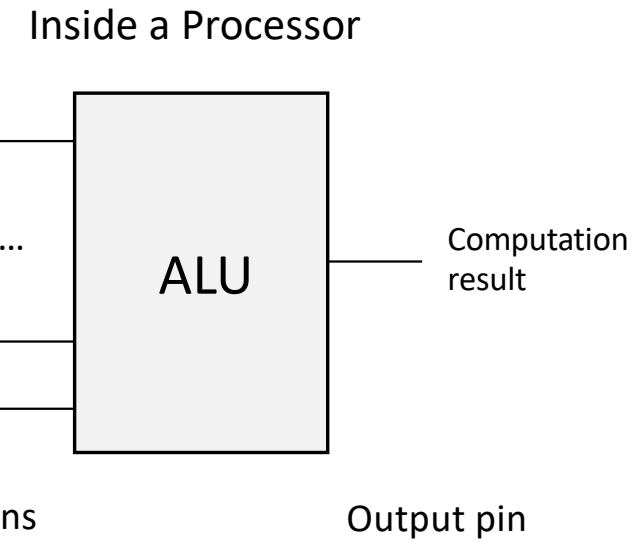
MIPS Instruction Set

Artem Burmyakov, Alexander Tormasov

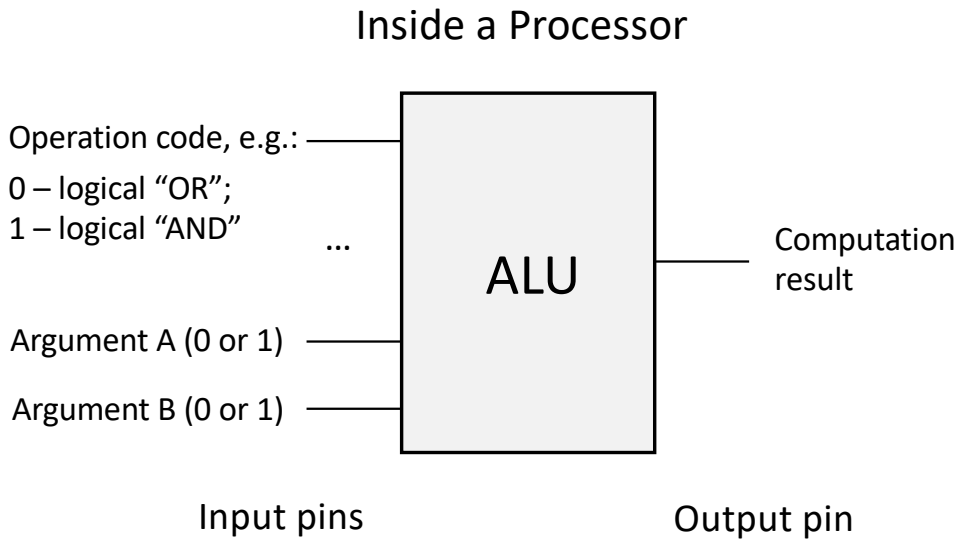
September 30, 2021

Innopolis University

Machine Language

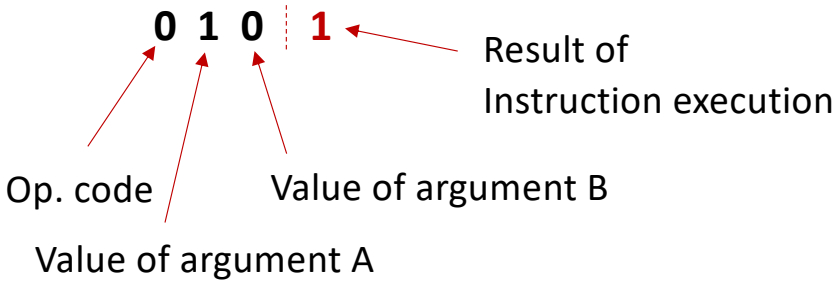


Machine Language

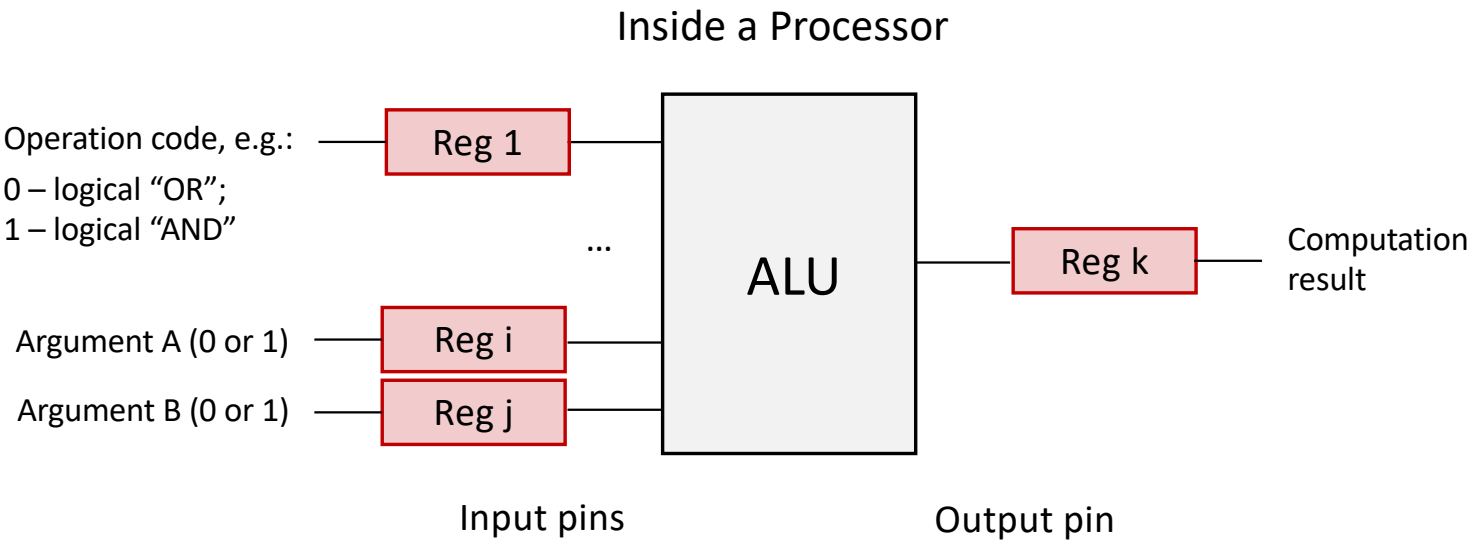


Format of a simple machine instruction (position matters): **<Operation code> <Value A> <Value B>**

Sample machine instruction (in binary code):

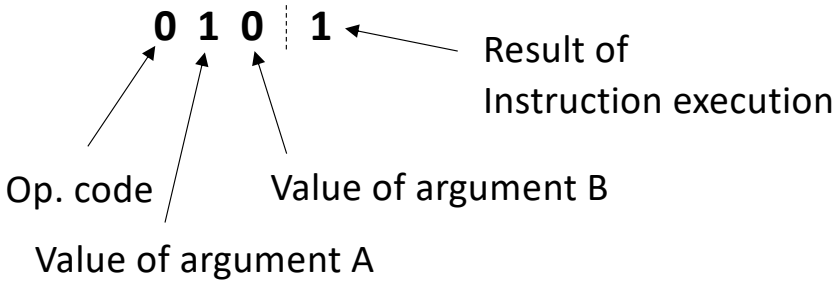


Registers – Solution for Synchronization Problem

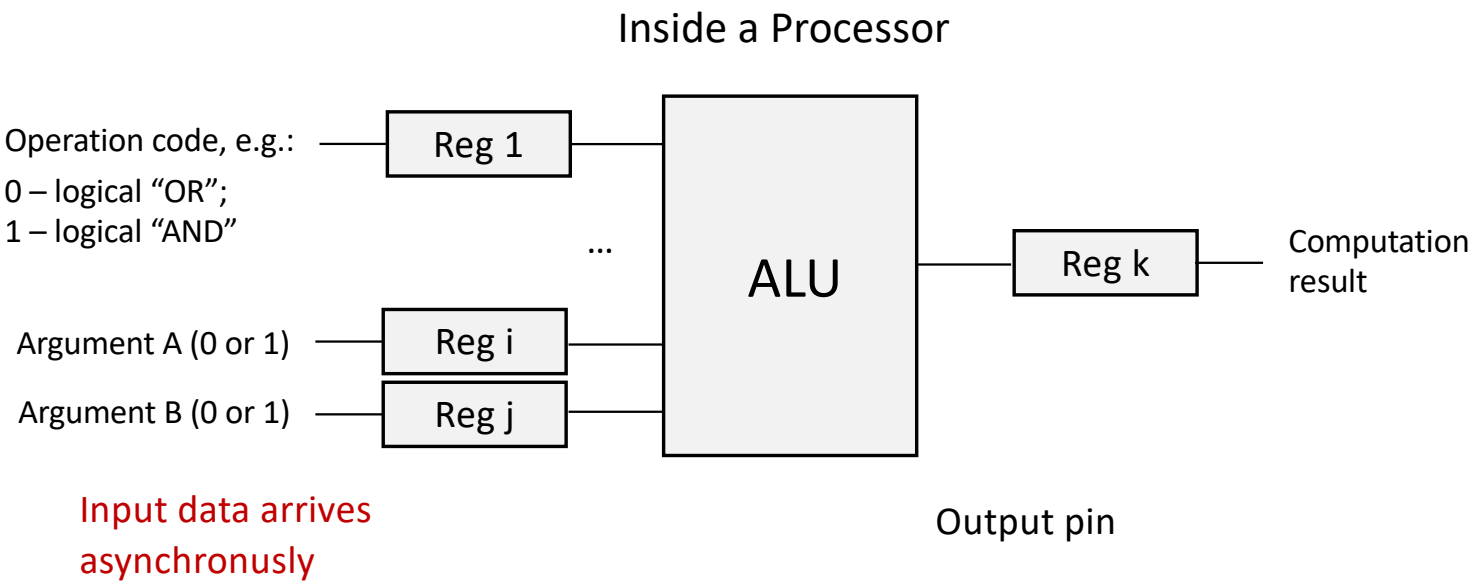


Format of a simple machine instruction (position matters): **<Operation code> <Value A> <Value B>**

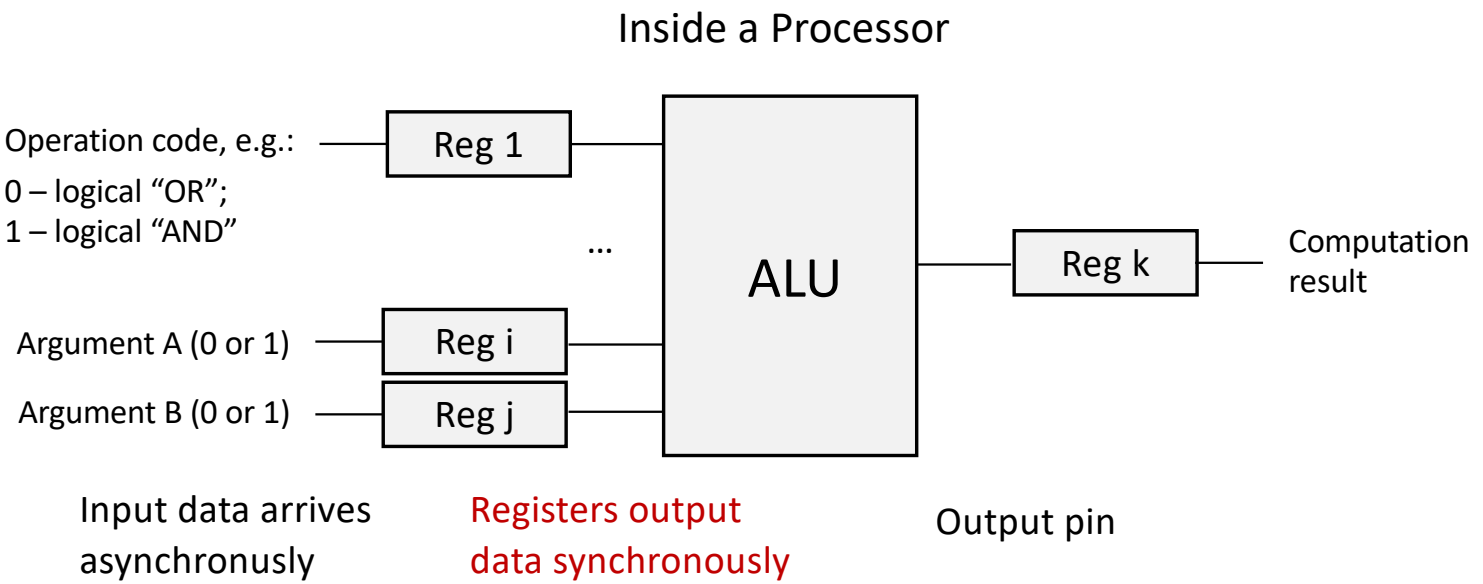
Sample machine instruction (in binary code):



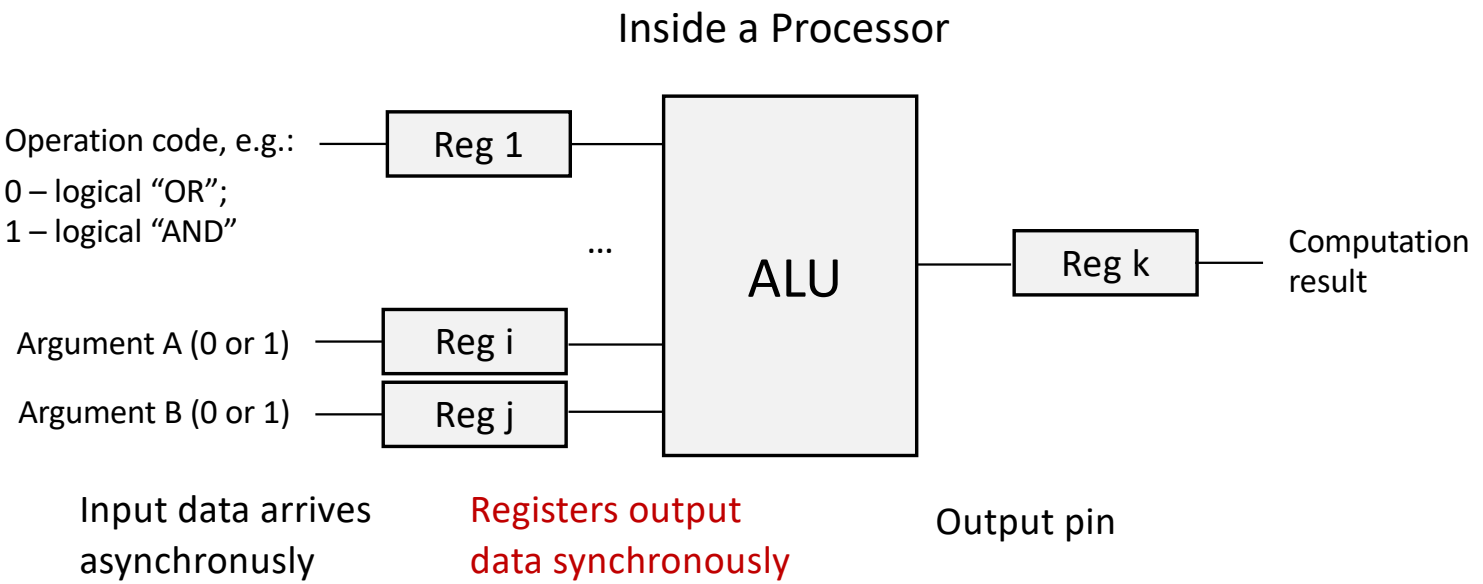
Registers – Solution for Synchronization Problem



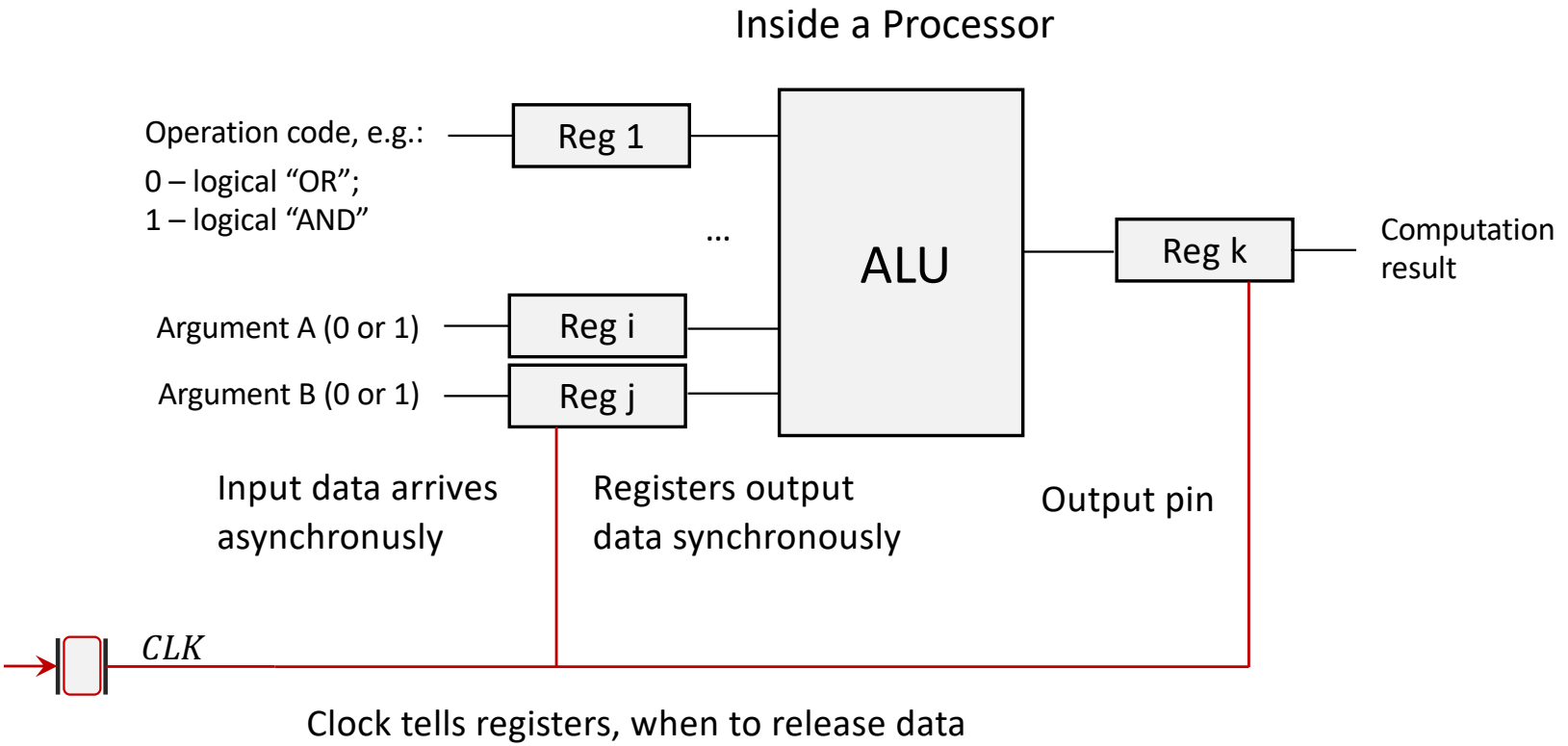
Registers – Solution for Synchronization Problem



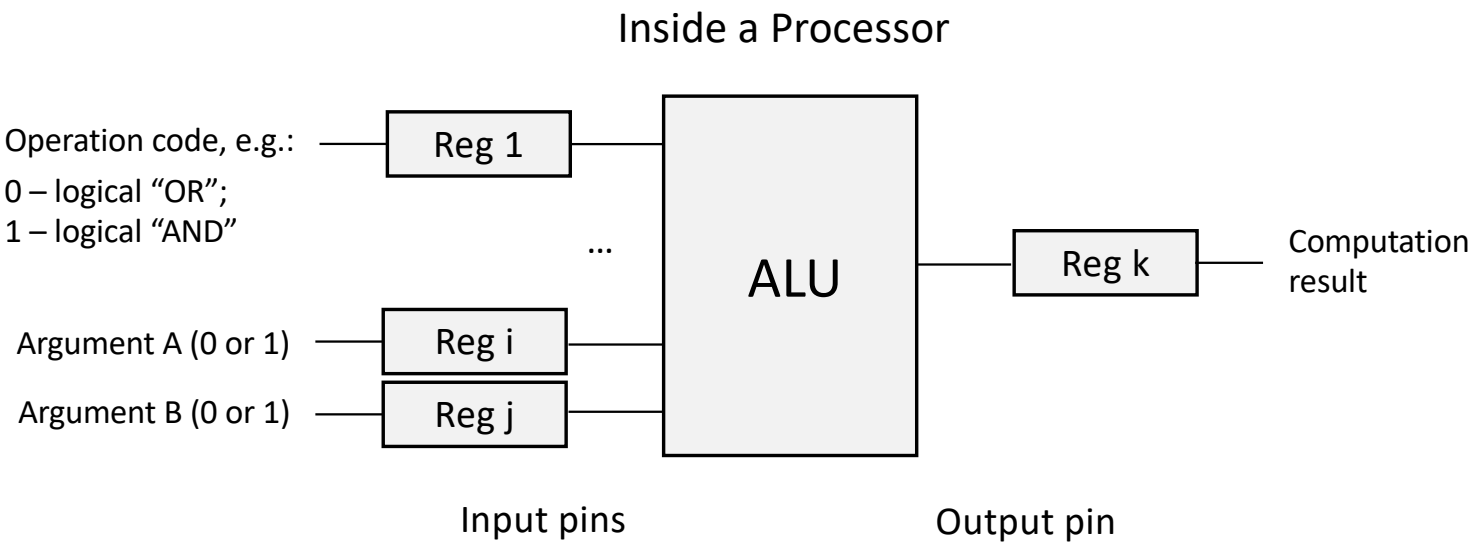
Registers – Solution for Synchronization Problem



Registers – Solution for Synchronization Problem

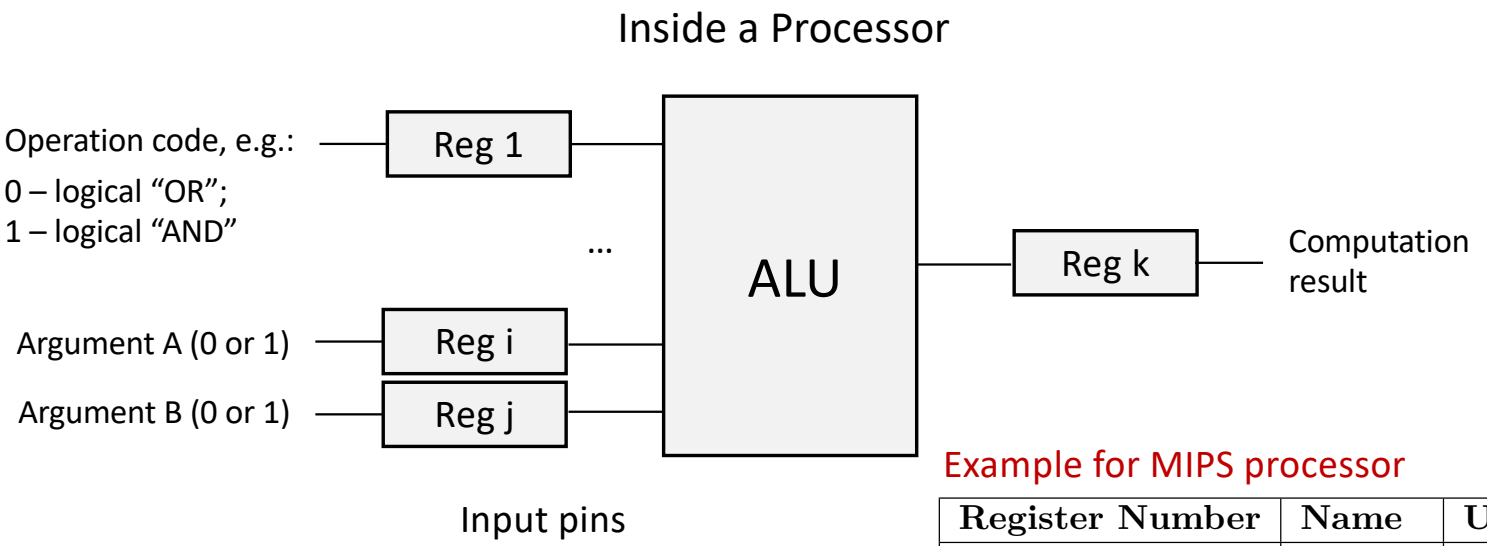


Registers



- *Each register has a name and number;*
- *Each register has a strictly defined purpose, e.g. for operation code*

Registers

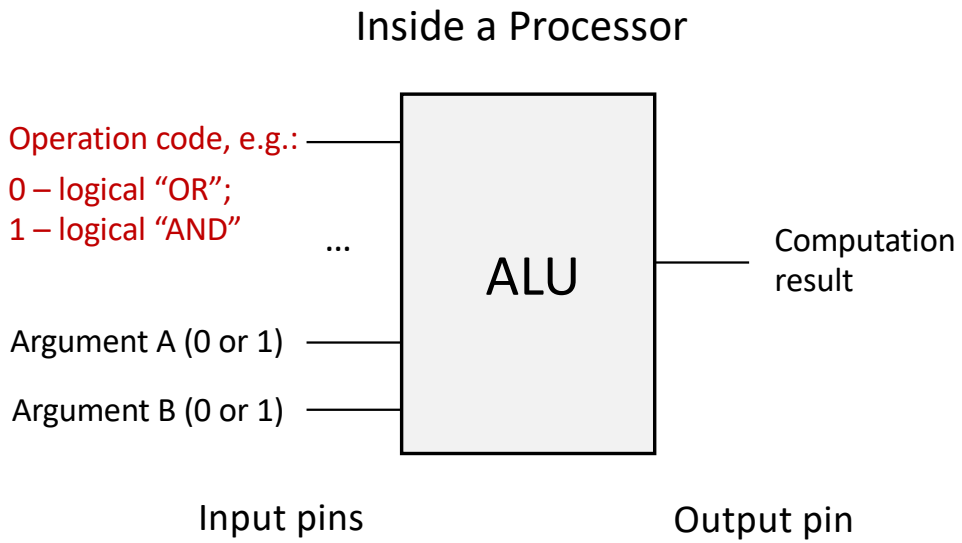


- *Each register has a name and number;*
- *Each register has a strictly defined purpose, e.g. for operation code*

Example for MIPS processor

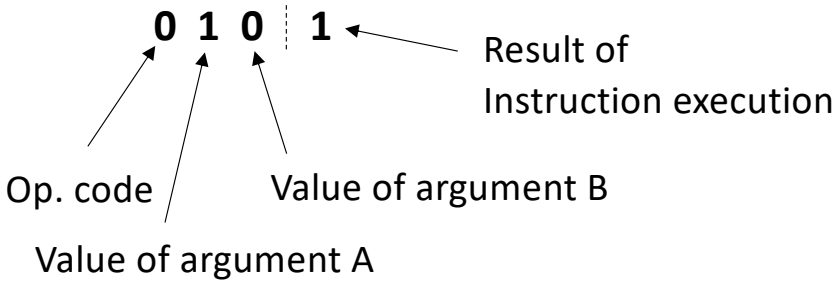
Register Number	Name	Usage
$0	$zero	constant
$1	$at	assembler temporary
$2–$3	$v0–$v1	function return values
$4–$7	$a0–$a3	function arguments
$8–$15	$t0–$t7	temporaries
$16–$23	$s0–$s7	saved temporaries
$24–$25	$t8–$t9	more temporaries
$26–$27	$k0–$k1	reserved for OS kernel
$28	$gp	global pointer
$29	$sp	stack pointer
$30	$fp	frame pointer
$31	$ra	return address

Machine Language



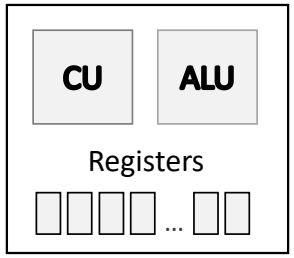
Format of a simple machine instruction (position matters): **<Operation code> <Value A> <Value B>**

Sample machine instruction (in binary code):



Instruction Set (IS) – the set of all instructions supported by a given processor

Instruction Set Example: MIPS



$s0-$s7
$t0-$t9
…

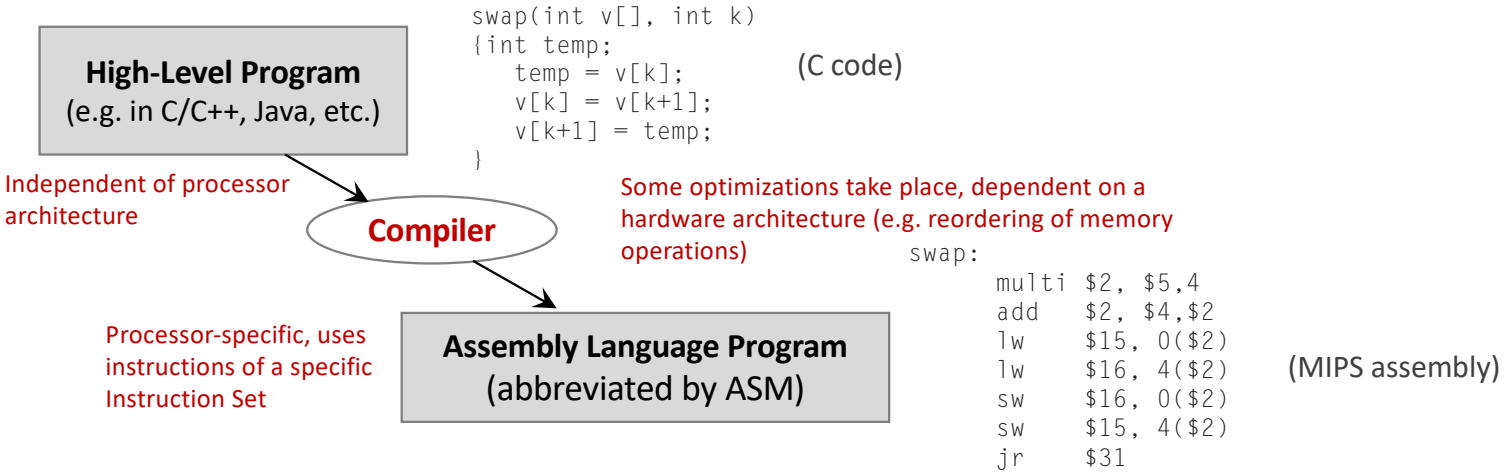
Names of registers

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Three register operands
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Three register operands
	add immediate	`addi $s1,$s2,20`	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	`lw $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	`sw $s1,20($s2)`	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	`lh $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	`lhu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	`sh $s1,20($s2)`	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	`lb $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	`lbu $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	`sb $s1,20($s2)`	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	`ll $s1,20($s2)`	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	`sc $s1,20($s2)`	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	`lui $s1,20`	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	`and $s1,$s2,$s3`	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	`or $s1,$s2,$s3`	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	`nor $s1,$s2,$s3`	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	`andi $s1,$s2,20`	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	`ori $s1,$s2,20`	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	`sll $s1,$s2,10`	$s1 = $s2 << 10	Shift left by constant
	shift right logical	`srl $s1,$s2,10`	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	`beq $s1,$s2,25`	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1,$s2,25`	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for `beq`, `bne`
	set on less than unsigned	`sltu $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	`slti $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	`sltiu $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to $ra	For switch, procedure return
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000	For procedure call

The reference is taken from:
David Patterson, John Hennessy;
Computer Organization and Design: The Hardware/Software Interface

Translation Hierarchy of a High-Level Program into Machine (Binary) Code



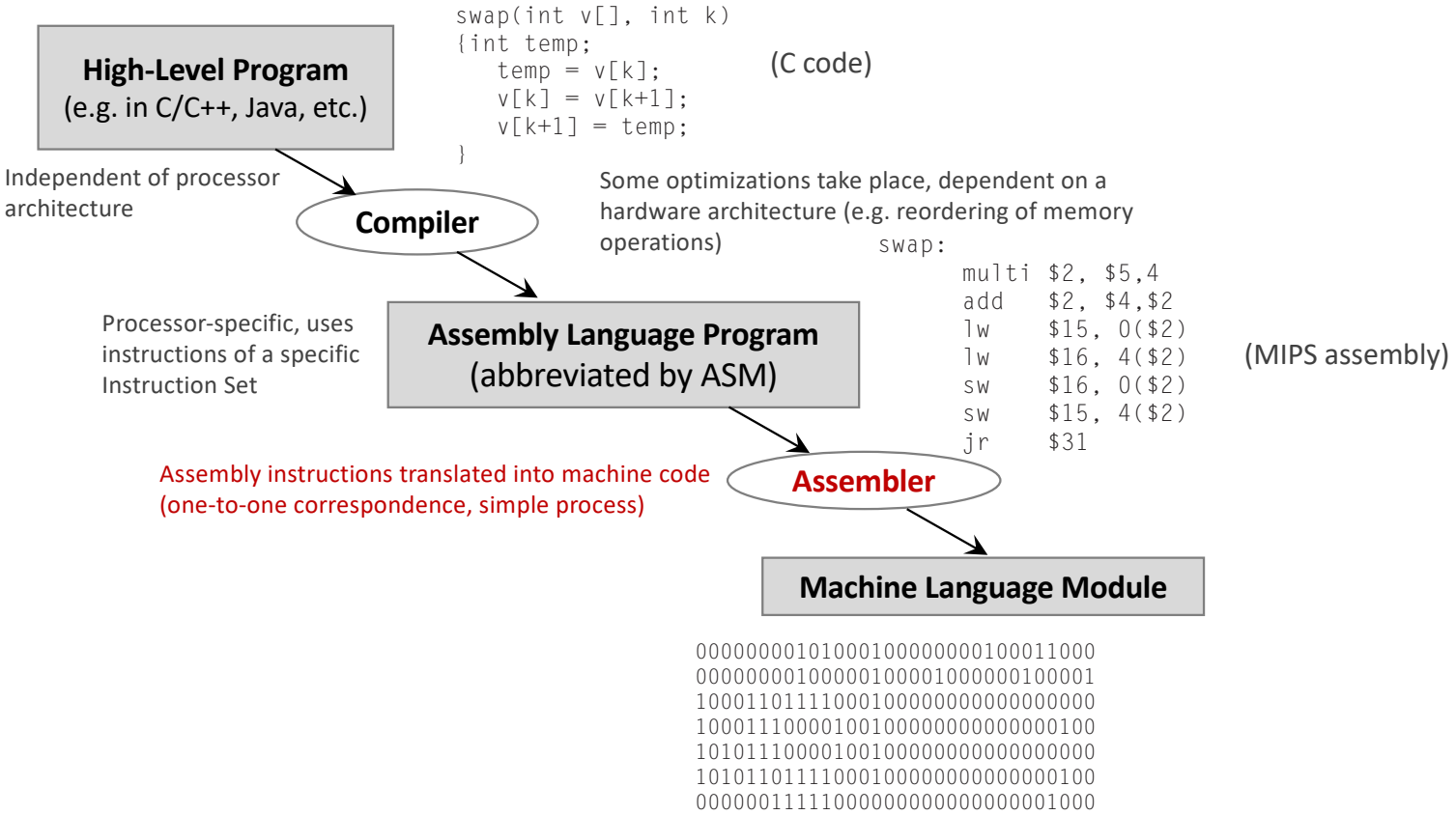

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

(C code)

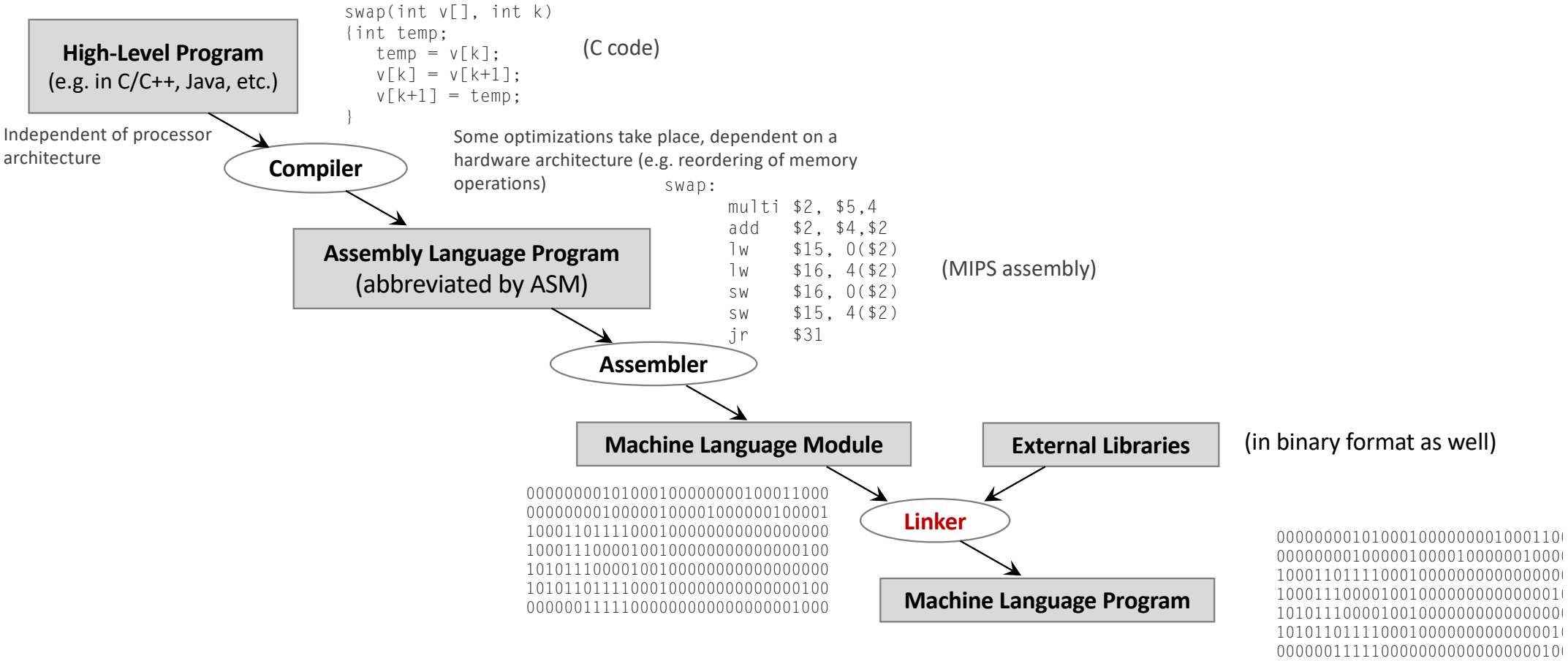
```
swap:
        multi $2, $5,4
        add   $2, $4,$2
        lw    $15, 0($2)
        lw    $16, 4($2)
        sw    $16, 0($2)
        sw    $15, 4($2)
        jr    $31
```

(MIPS assembly)

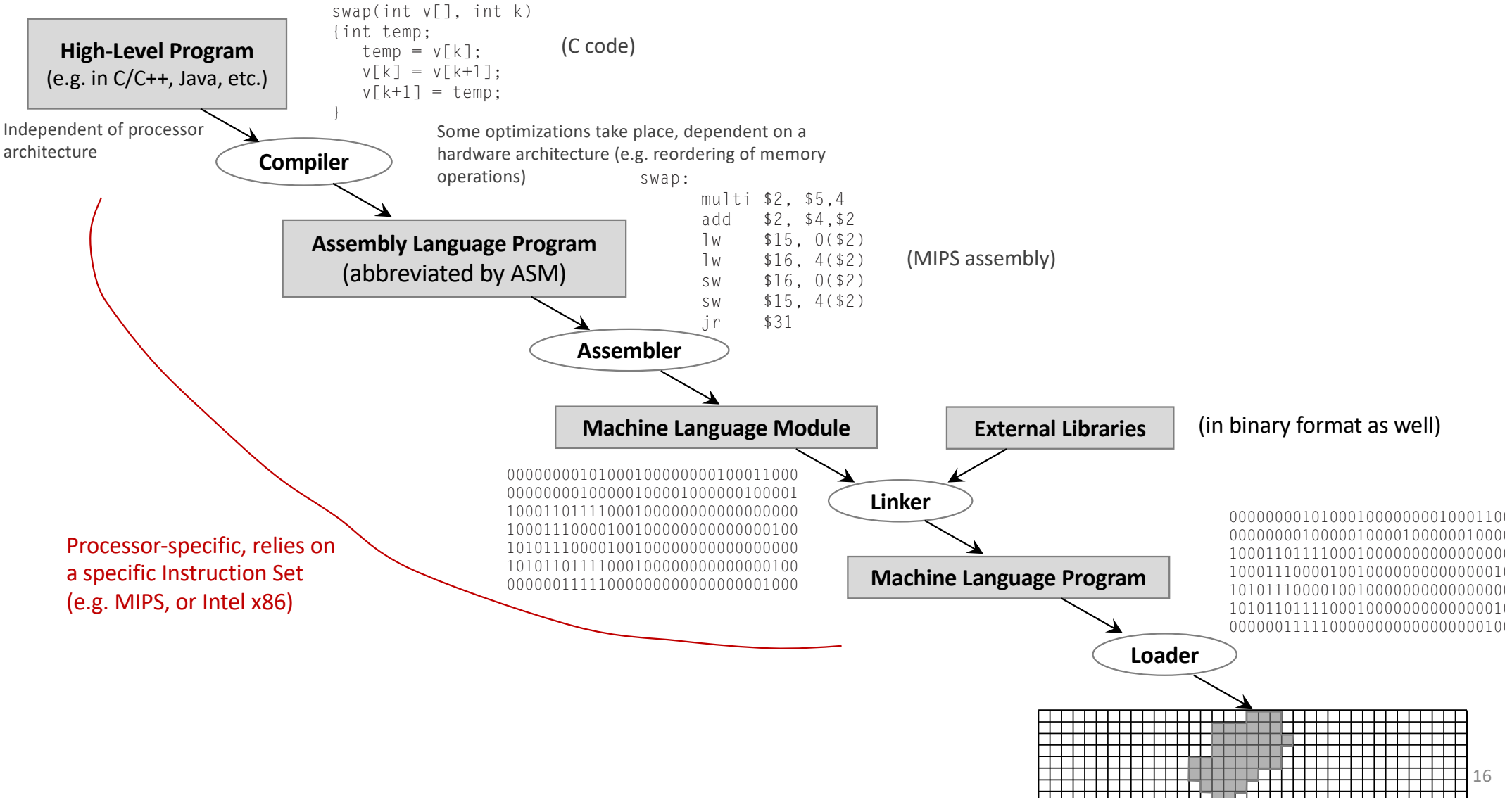
Translation Hierarchy of a High-Level Program into Machine (Binary) Code



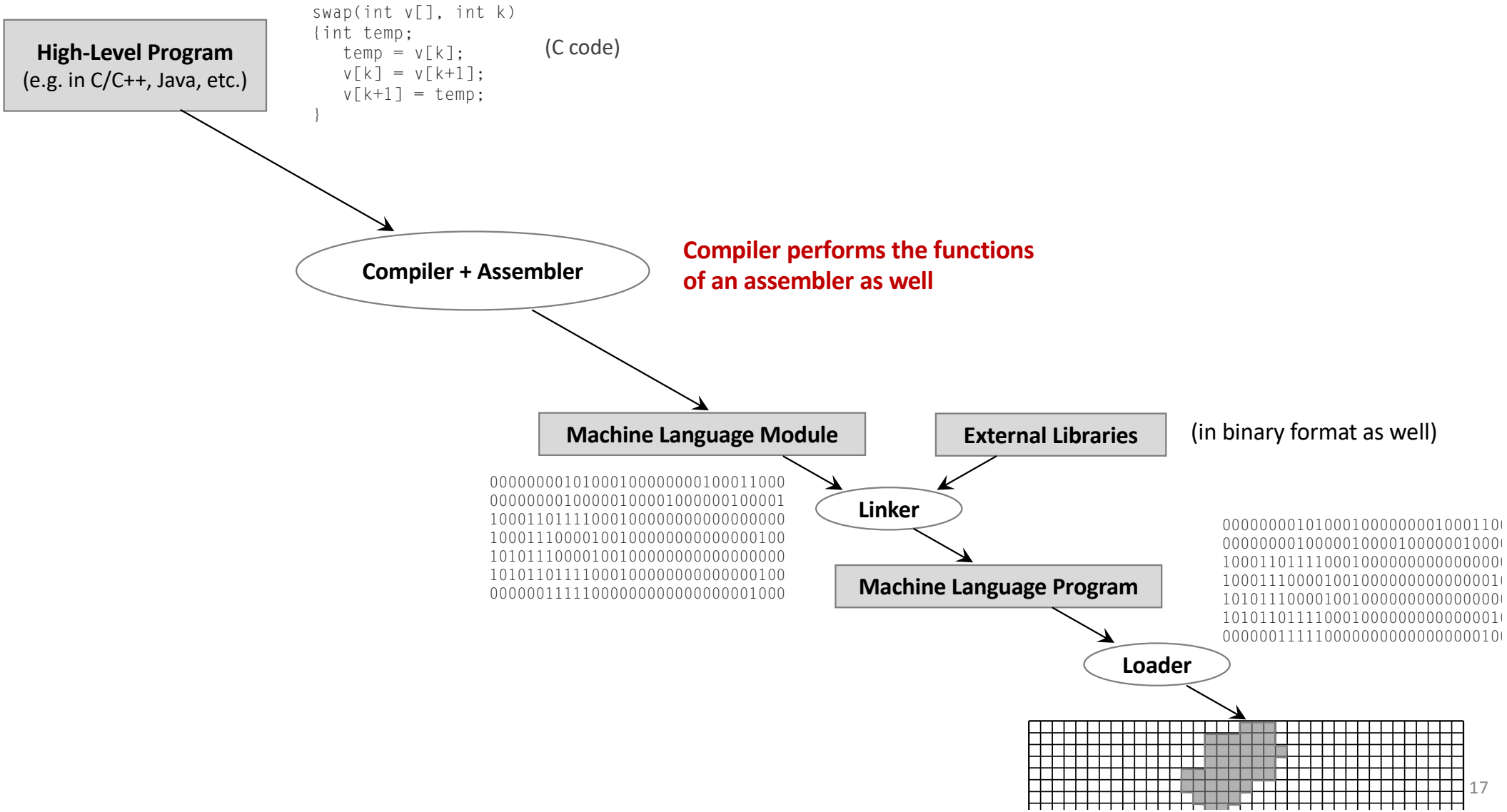
Translation Hierarchy of a High-Level Program into Machine (Binary) Code



Translation Hierarchy of a High-Level Program into Machine (Binary) Code



Variations of a Translation Hierarchy



Program Translation: From C to MIPS Binary

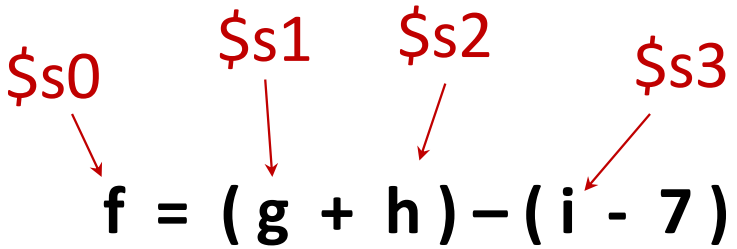
C program instruction to be translated:

f = (g + h) – (i - 7)

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively



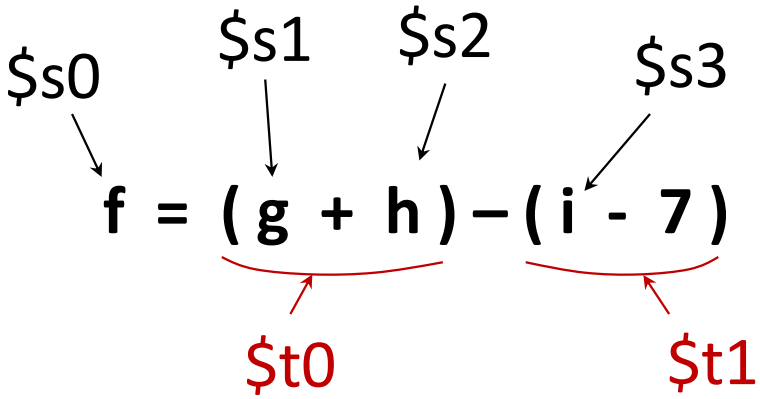
Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results



Program Translation: From C to MIPS Binary

Step 1:

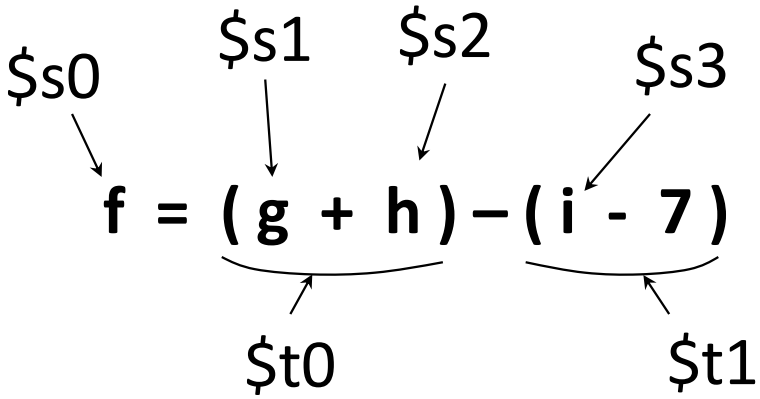
Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

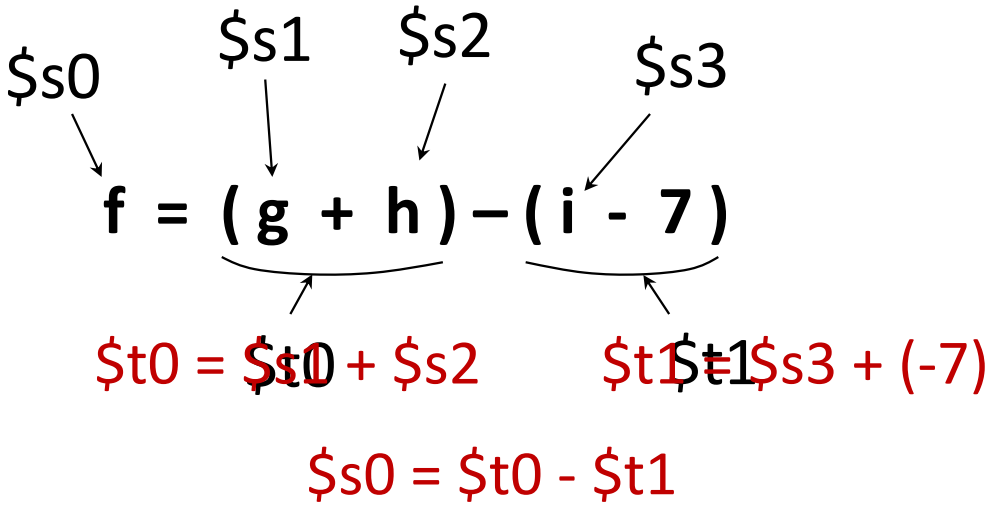
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2
addi  $t1,  $s3,  -7
sub   $s0,  $t0,  $t1
```



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

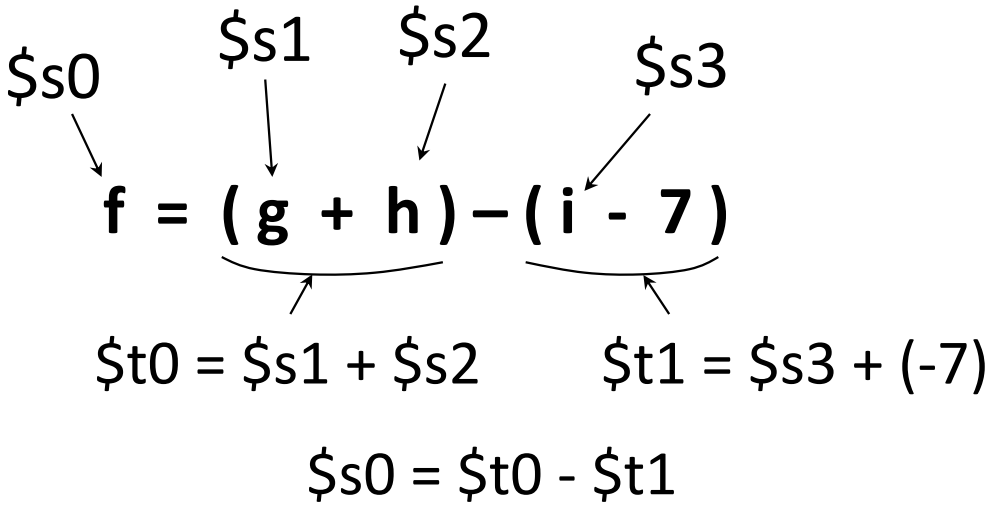
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```

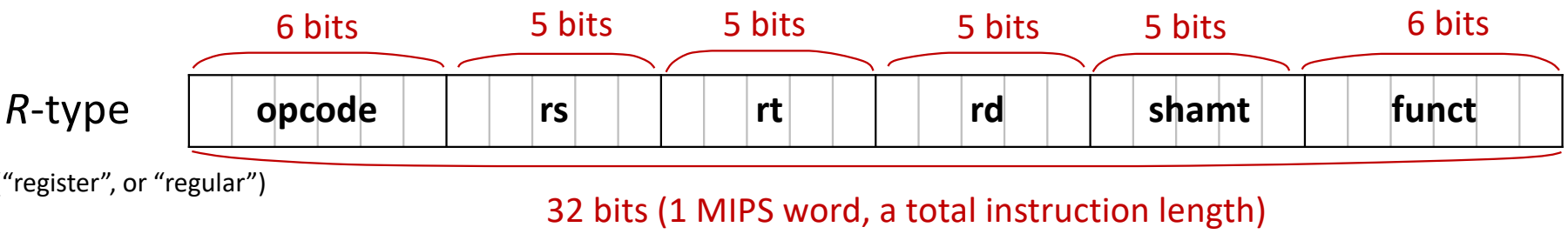


Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes

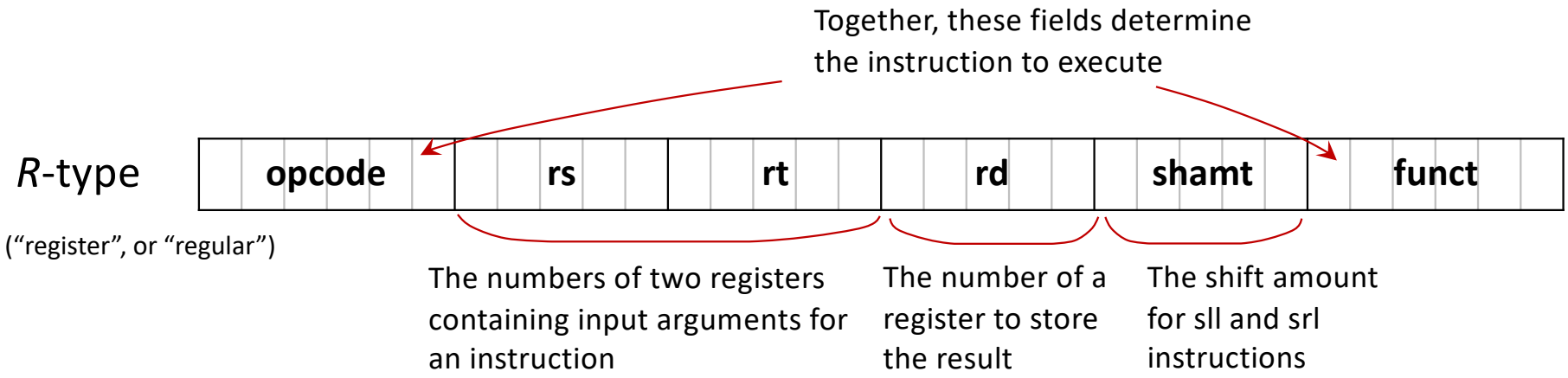
Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



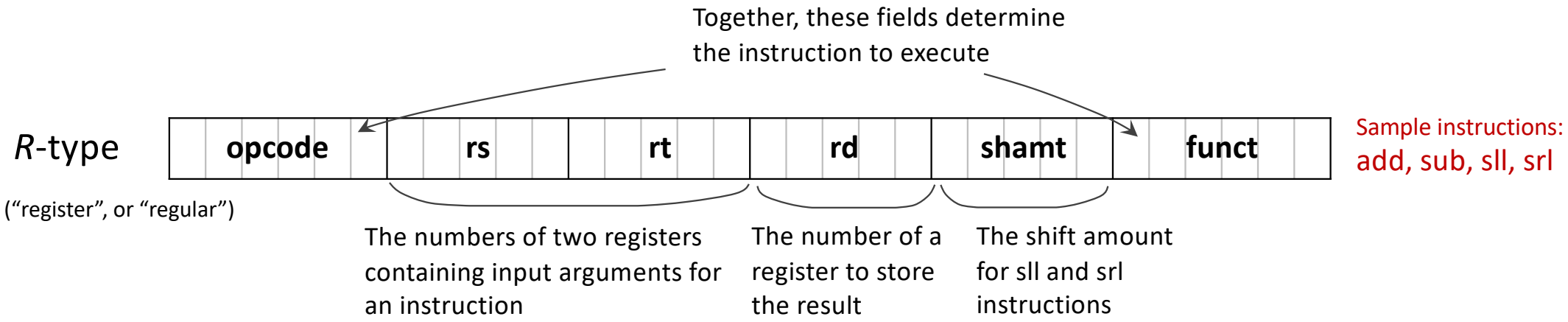
Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



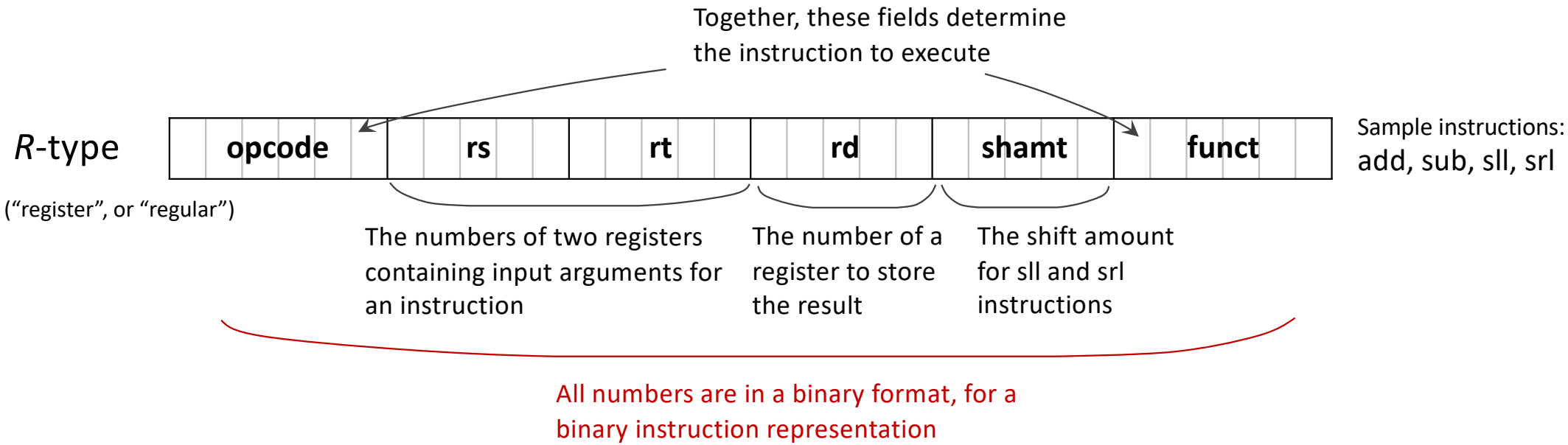
Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes

R-type

add, sub, sll, srl

("register", or "regular")

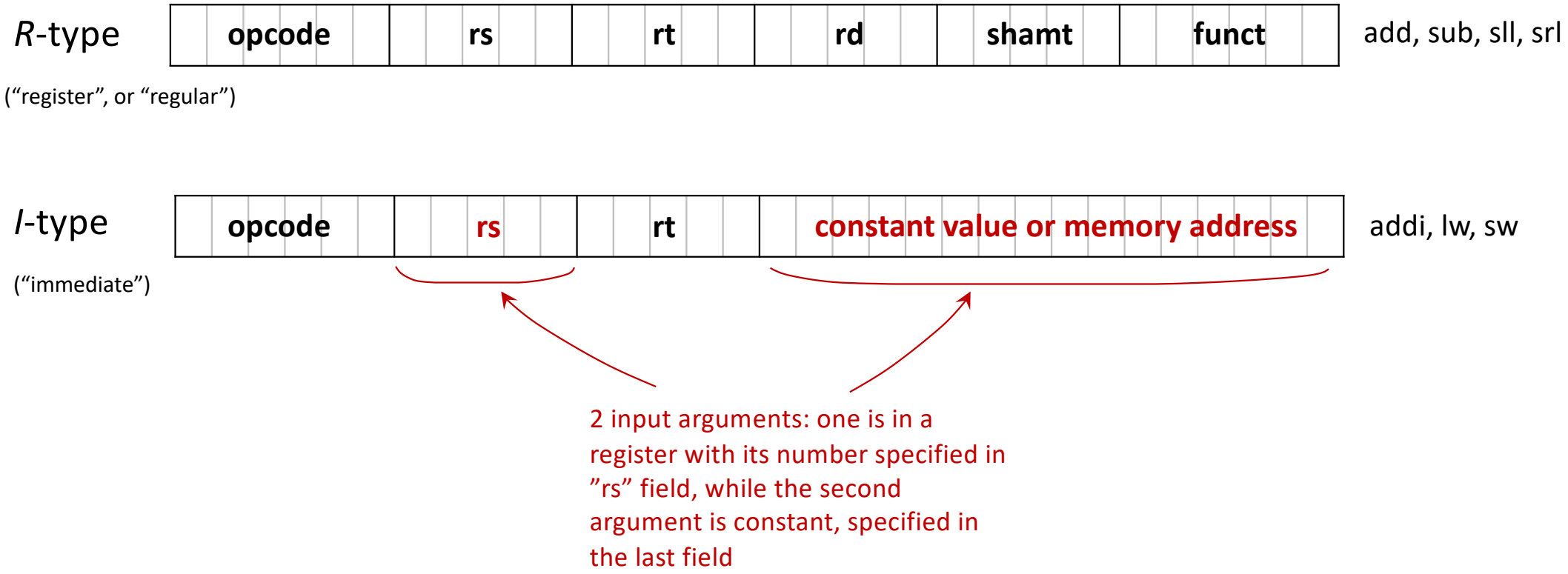
I-type

addi, lw, sw

("immediate")

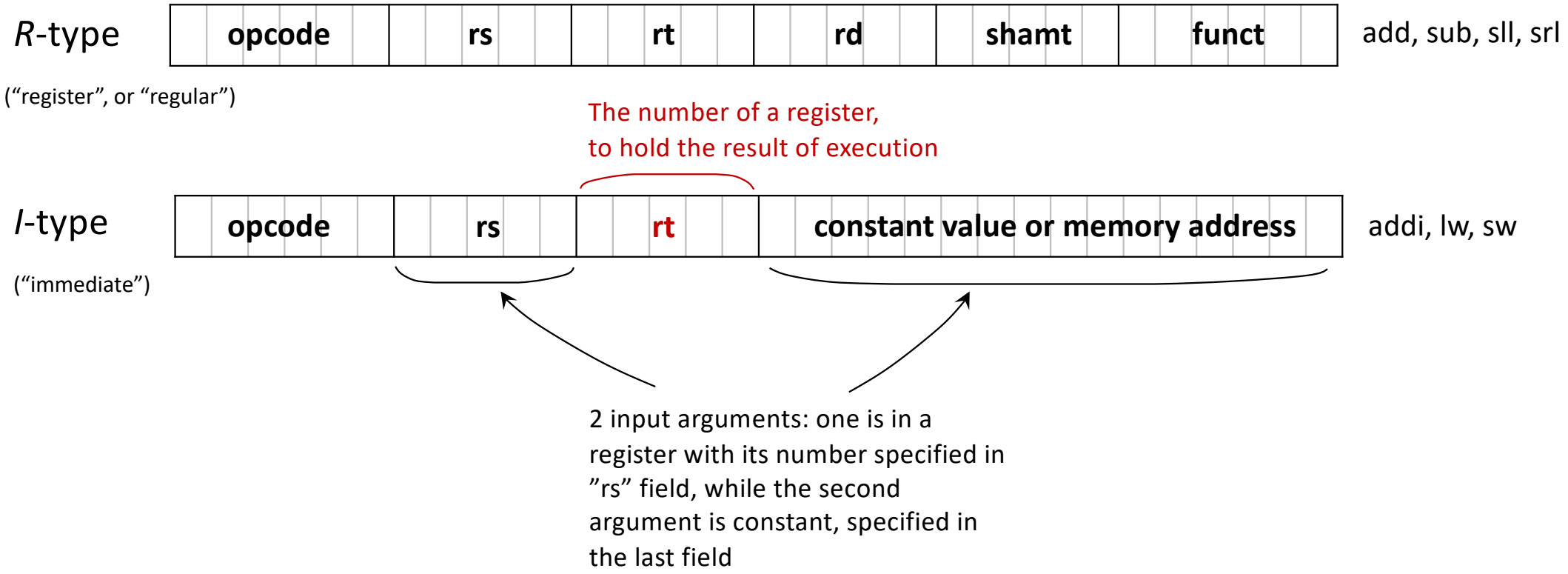
Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes



Types of MIPS Processor Instructions: R, I, and J

All MIPS instructions are 32 bit long in their binary representation;
The difference is in the number of fields, their meaning, and sizes

R-type

add, sub, sll, srl

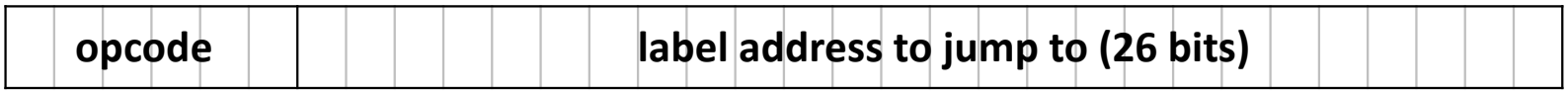
("register", or "regular")

I-type

addi, lw, sw

("immediate")

J-type



j, jr, jal

("jump")

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:

opcode	rs	rt	rd	shamt	funct

(6 fields, R-type)

opcode	rs	rt	constant or address

(3 fields, I-type)

opcode	rs	rt	rd	shamt	funct

(6 fields, R-type)

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:

add $t0, $s1, $s2

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

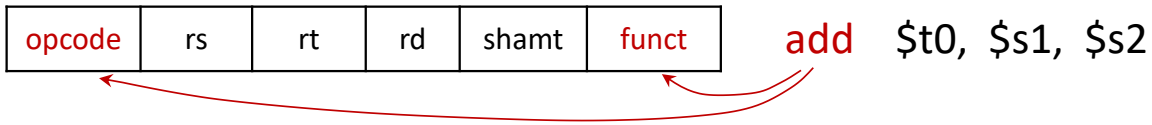
```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

MIPS instruction	Opcode	Funct
add	0	32
sub	0	34
addi	8	N/A

Step 4:

Convert MIPS instructions into decimal format:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

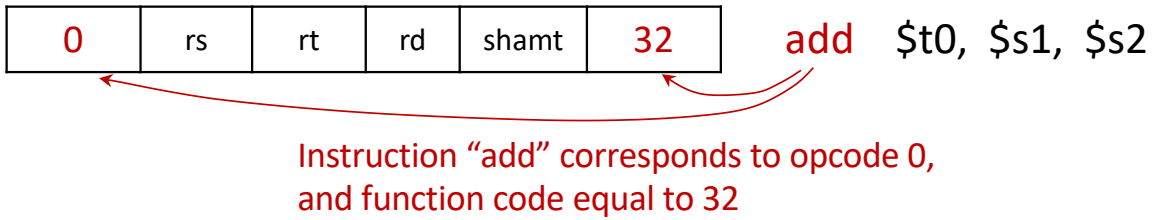
```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

MIPS instruction	Opcode	Funct
add	0	32
sub	0	34
addi	8	N/A

Step 4:

Convert MIPS instructions into decimal format:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

MIPS instruction	Opcode	Funct
add	0	32
sub	0	34
addi	8	N/A

I-type instructions do not have "funct" field, but differ in opcode values

All *R*-type instructions share opcode "0", but differ in function codes ("funct")

Step 4:

Convert MIPS instructions into decimal format:

0	rs	rt	rd	shamt	32

add $t0, $s1, $s2

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

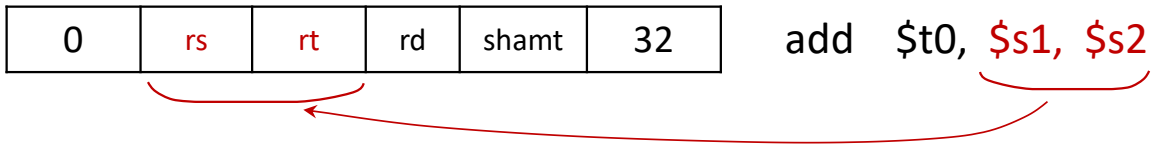
Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2      (R-type)
addi  $t1, $s3, -7       (I-type)
sub   $s0, $t0, $t1      (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

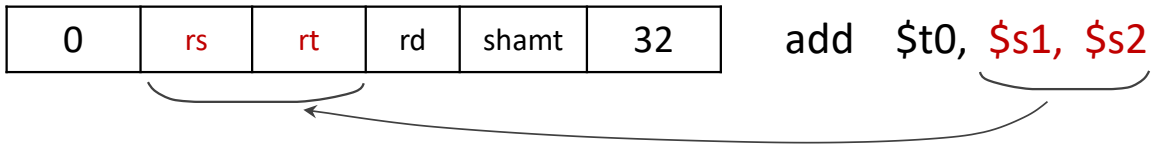
From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

…

Step 4:

Convert MIPS instructions into decimal format:



Place the numbers of registers $s1 and $s2 into fields “rs” and “rt”, respectively; these numbers are taken from MIPS specification

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

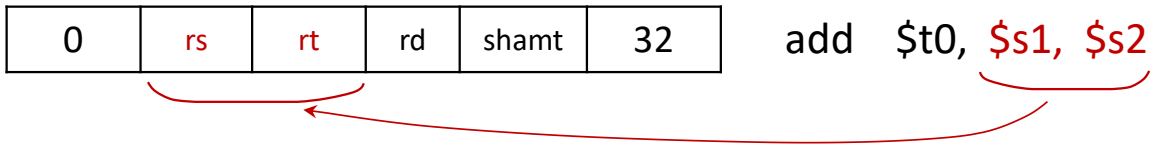
```
add   $t0, $s1, $s2      (R-type)
addi  $t1, $s3, -7       (I-type)
sub   $s0, $t0, $t1      (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

Step 4:

Convert MIPS instructions into decimal format:



Registers $s1 and $s2 correspond to numbers 17 and 18, respectively

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

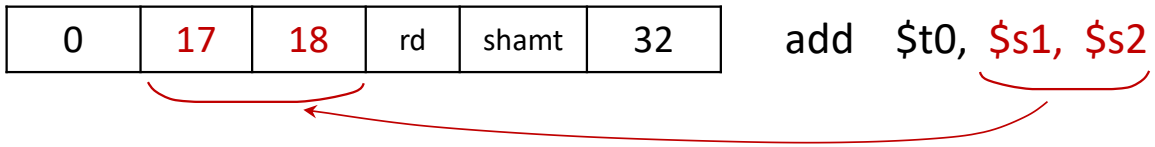
```
add   $t0, $s1, $s2     (R-type)
addi  $t1, $s3, -7      (I-type)
sub   $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

Step 4:

Convert MIPS instructions into decimal format:



Registers $s1 and $s2 correspond to numbers 17 and 18, respectively

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

Step 4:

Convert MIPS instructions into decimal format:

add $t0, $s1, $s2

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

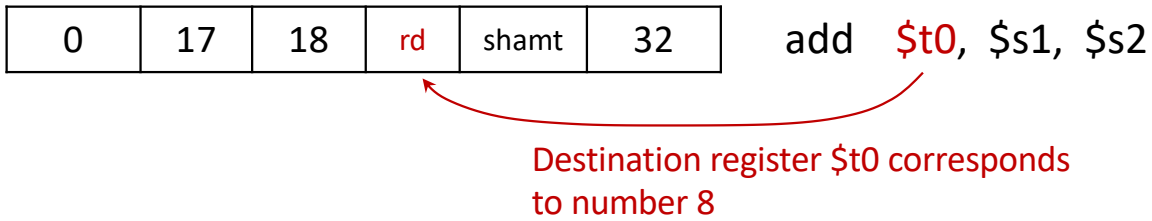
```
add  $t0, $s1, $s2    (R-type)
addi $t1, $s3, -7     (I-type)
sub  $s0, $t0, $t1    (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

Step 4:

Convert MIPS instructions into decimal format:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

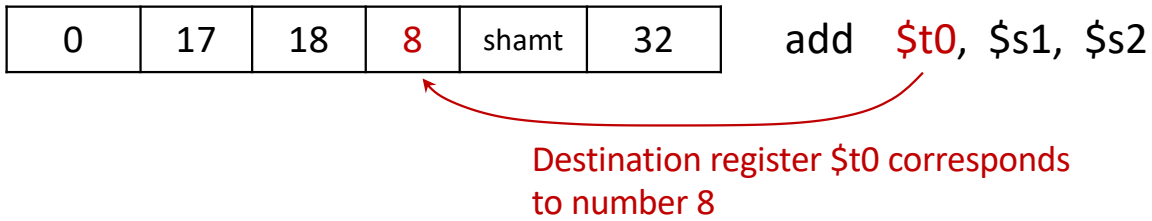
```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

Step 4:

Convert MIPS instructions into decimal format:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2      (R-type)
addi  $t1, $s3, -7       (I-type)
sub   $s0, $t0, $t1      (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	shamt	32

add $t0, $s1, $s2

Whenever field “shamt” is unused, it is reset to 0

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32

add $t0, $s1, $s2

Whenever field “shamt” is unused, it is reset to 0

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```

Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
opcode	rs	rt	constant or address		
opcode	rs	rt	rd	shamt	funct

```
add  $t0, $s1, $s2
addi $t1, $s3, -7
sub  $s0, $t0, $t1
```

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2    (R-type)
addi $t1, $s3, -7     (I-type)
sub  $s0, $t0, $t1    (R-type)
```

From MIPS processor specification:

Register Name	Register Number
$t0	8
$t1	9
$s0	16
$s1	17
$s2	18
$s3	19

…

MIPS instruction	Opcode	Funct
add	0	32
sub	0	34
addi	8	N/A

…

Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	8	9	16	0	34

```
add  $t0, $s1, $s2
addi $t1, $s3, -7
sub  $s0, $t0, $t1
```

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

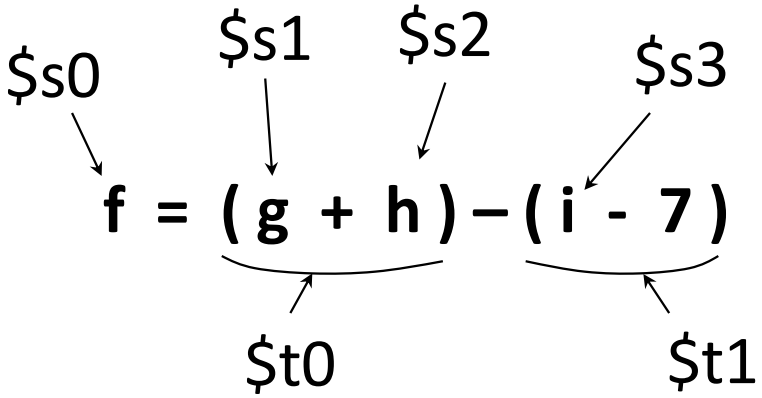
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	8	9	16	0	34

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

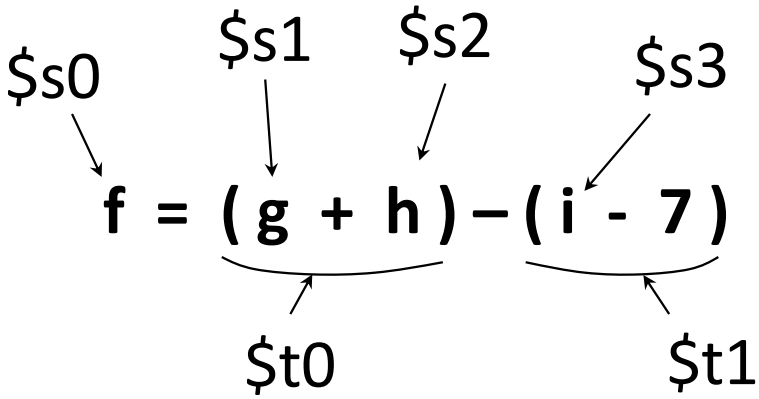
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add  $t0, $s1, $s2     (R-type)
addi $t1, $s3, -7      (I-type)
sub  $s0, $t0, $t1     (R-type)
```



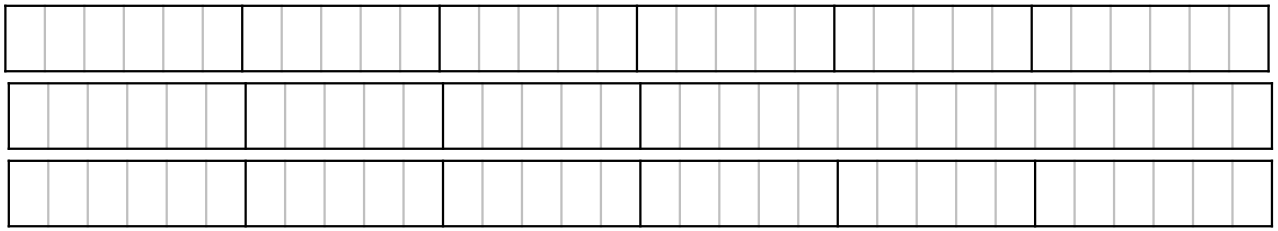
Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	8	9	16	0	34

Step 5:

Convert instructions from decimal into binary representation:



Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

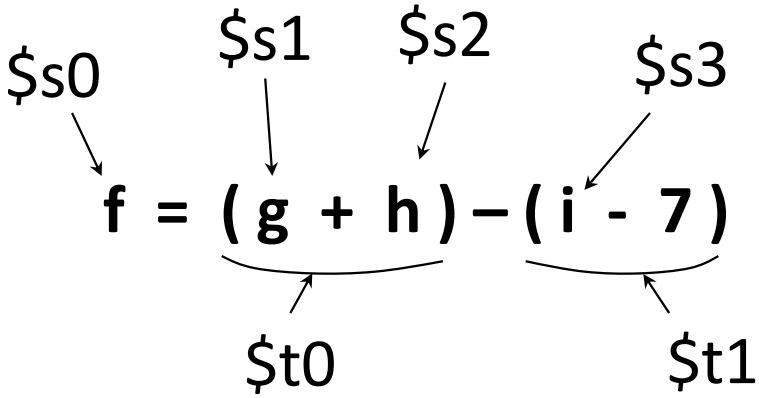
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2     (R-type)
addi  $t1, $s3, -7      (I-type)
sub   $s0, $t0, $t1     (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32

Step 5:

Convert instructions from decimal into binary representation:

0 0 0 0 0 0																															

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

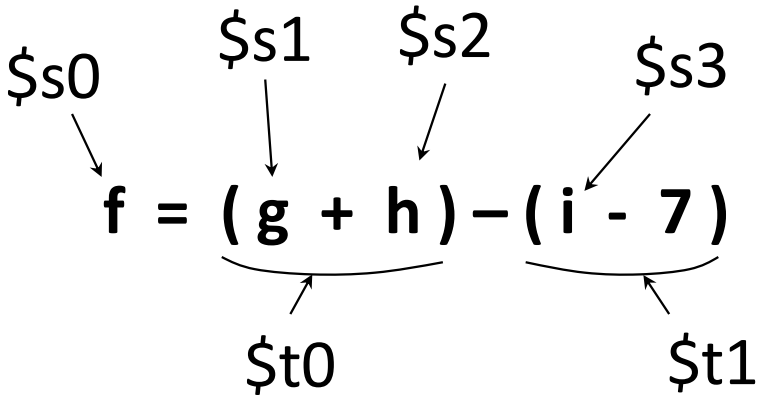
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2      (R-type)
addi  $t1, $s3, -7       (I-type)
sub   $s0, $t0, $t1      (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0	1	0	0	0	1																					

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

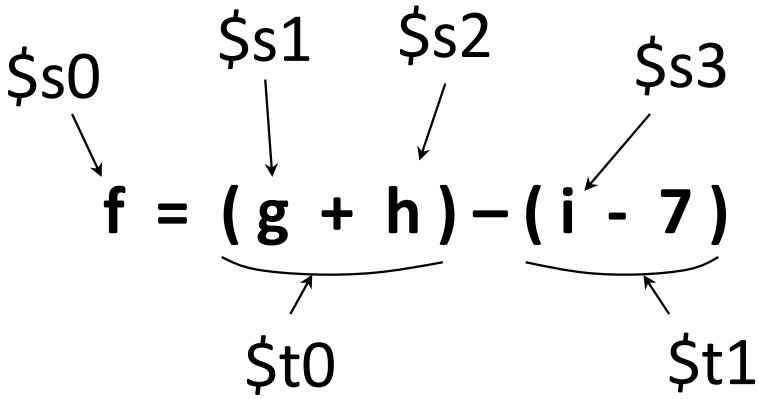
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2      (R-type)
addi  $t1, $s3, -7       (I-type)
sub   $s0, $t0, $t1      (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

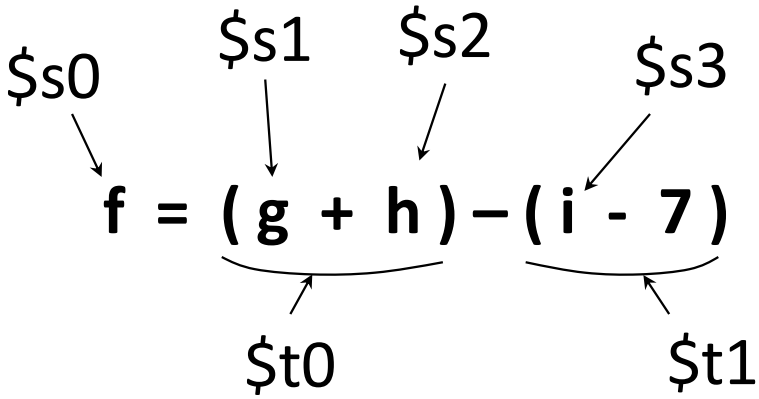
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	8	9	16	0	34

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	
0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

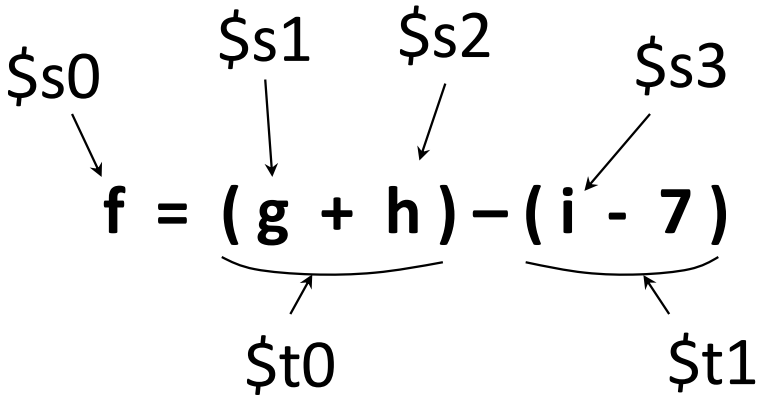
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0, $s1, $s2     (R-type)
addi  $t1, $s3, -7      (I-type)
sub   $s0, $t0, $t1     (R-type)
```



$(-7)_{\text{decimal}} = (1111111111111001)_{\text{binary}}$

Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	8	9	16	0	34

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	
0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

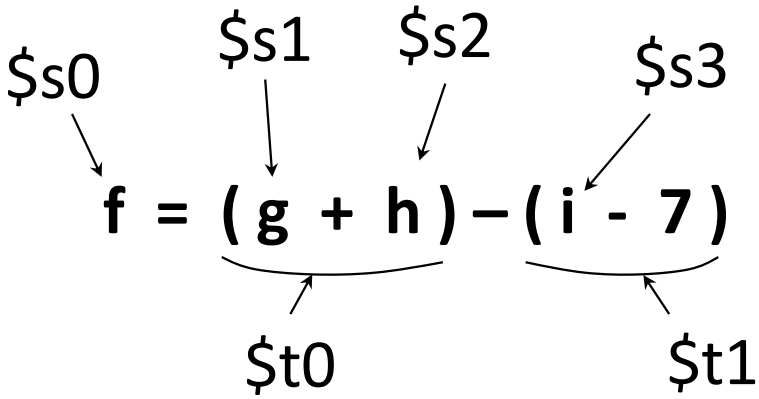
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	8	9	16	0	34

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	
0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0

Program Translation: From C to MIPS Binary

Step 1:

Assign registers, e.g. $s0,…, $s3, to program variables f, g, h, and i, respectively

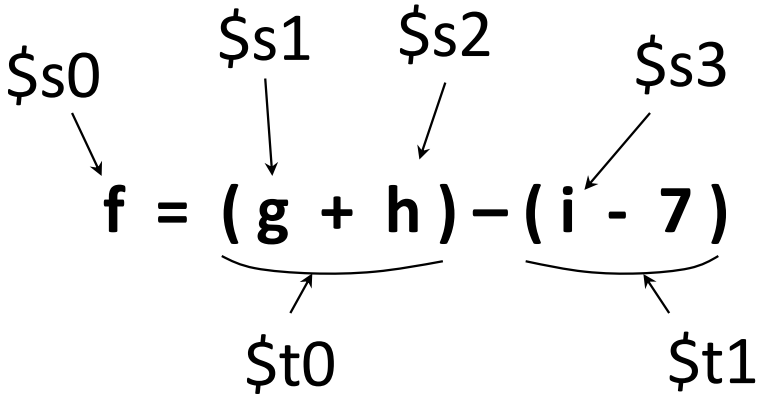
Step 2:

Assign temporary registers $t0, $t1 to hold temporary results

Step 3:

Convert the C instruction on the right into the following 3 MIPS instructions:

```
add   $t0,  $s1,  $s2      (R-type)
addi  $t1,  $s3,  -7       (I-type)
sub   $s0,  $t0,  $t1      (R-type)
```



Step 4:

Convert MIPS instructions into decimal format:

0	17	18	8	0	32
8	19	9	-7		
0	8	9	16	0	34

Step 5:

Convert instructions from decimal into binary representation:

0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0

An Overview of a Program Execution Chain