

CSE3038 PS - Chapter 2

Lokman ALTIN
lokman.altin@marmara.edu.tr

Patterson and Hennessy, *Computer Organization and Design, 4th and 5th edition*

Problem 1

The following problems deal with translating from C to MIPS. Assume that the variables *f*, *g*, *h*, *i*, and *j* are assigned to registers *\$s0*, *\$s1*, *\$s2*, *\$s3*, and *\$s4*, respectively. Assume that the base address of the arrays *A* and *B* are in registers *\$s6* and *\$s7*, respectively.

a.	<code>f = g + h + B[4];</code>
b.	<code>f = g - A[B[4]];</code>

For the C statements above, what is the corresponding MIPS assembly code?

Problem 1

a) $f = g + h + B[4]$

- $f \rightarrow \$s0, \quad g \rightarrow \$s1, \quad h \rightarrow \$s2, \quad B \rightarrow \$s7$

add	\$t0, \$s1, \$s2	# \$t0 = g + h
lw	\$t1, 16(\$s7)	# \$t1 = B[4]
add	\$s0, \$t0, \$t1	# f = g + h + B[4]

Problem 1

b) $f = g - A[B[4]]$

- $f \rightarrow \$s0, \quad g \rightarrow \$s1, \quad A \rightarrow \$s6, \quad B \rightarrow \$s7$

lw	\$t0, 16(\$s7)	# \$t0 = B[4]
sll	\$t1, \$t0, 2	# \$t1 = B[4] * 4
add	\$t2, \$s6, \$t1	# \$t2 = A + (B[4] * 4)
lw	\$t3, 0(\$t2)	# \$t3 = A[B[4]]
sub	\$s0, \$s1, \$t3	# f = g - A[B[4]]

Problem 2

In the following problems, we will be investigating memory operations in the context of a MIPS processor. The table below shows the values of an array stored in memory.

a.	Address	Data
	12	1
	8	6
	4	4
	0	2
b.	Address	Data
	16	1
	12	2
	8	3
	4	4
	0	5

For the memory locations in the table above, write MIPS code to sort the data from lowest to highest, placing the lowest value in the smallest memory location. Use a minimum number of MIPS instructions. Assume the base address of Array is stored in register \$s6.

Problem 2.a

lw	\$t0, 0(\$s6)	# \$t0 = Array[0] = 2
lw	\$t1, 4(\$s6)	# \$t1 = Array[1] = 4
lw	\$t2, 8(\$s6)	# \$t2 = Array[2] = 6
lw	\$t3, 12(\$s6)	# \$t3 = Array[3] = 1
sw	\$t3, 0(\$s6)	# Array[0] = 1
sw	\$t0, 4(\$s6)	# Array[1] = 2
sw	\$t1, 8(\$s6)	# Array[2] = 4
sw	\$t2, 12(\$s6)	# Array[3] = 6

Problem 2.b

lw	\$t0, 0(\$s6)	# \$t0 = Array[0] = 5
lw	\$t1, 4(\$s6)	# \$t1 = Array[1] = 4
lw	\$t2, 12(\$s6)	# \$t2 = Array[3] = 2
lw	\$t3, 16(\$s6)	# \$t3 = Array[4] = 1
sw	\$t3, 0(\$s6)	# Array[0] = 1
sw	\$t2, 4(\$s6)	# Array[1] = 2
sw	\$t1, 12(\$s6)	# Array[3] = 4
sw	\$t0, 16(\$s6)	# Array[4] = 5

Array[2] = 3, already in place!

Problem 3.a.

For the MIPS assembly instructions below, what is the corresponding C statement? Assume that the variables **f**, **g**, **h**, **i**, and **j** are assigned to registers **\$s0**, **\$s1**, **\$s2**, **\$s3**, and **\$s4**, respectively. Assume that the base address of the arrays **A** and **B** are in registers **\$s6** and **\$s7**, respectively.

sll \$t0, \$s0, 2	# \$t0 = f * 4
add \$t0, \$s6, \$t0	# \$t0 = &A[f]
sll \$t1, \$s1, 2	# \$t1 = g * 4
add \$t1, \$s7, \$t1	# \$t1 = &B[g]
lw \$s0, 0(\$t0)	# f = A[f]
addi \$t2, \$t0, 4	
lw \$t0, 0(\$t2)	
add \$t0, \$t0, \$s0	
sw \$t0, 0(\$t1)	

Solution: $B[g] = A[f] + A[1+f];$

Problem 3.b.

For the MIPS assembly instructions in Problem 1.a, rewrite the assembly code to minimize the number of MIPS instructions (if possible) needed to carry out the same function.

Solution:

sll \$t0, \$s0, 2	# \$t0 = f * 4
add \$t0, \$s6, \$t0	# \$t0 = &A[f]
sll \$t1, \$s1, 2	# \$t1 = g * 4
add \$t1, \$s7, \$t1	# \$t1 = &B[g]
lw \$s0, 0(\$t0)	# f = A[f]
lw \$t0, 4(\$t0)	# t0 = A[f+1]
add \$t0, \$t0, \$s0	# t0 = A[f] + A[f+1]
sw \$t0, 0(\$t1)	# B[g] = t0

Problem 4

In the following problems, the data table contains bits that represent the opcode of an instruction. You will be asked to translate the entries into assembly code and determine what format of MIPS instruction the bits represent.

a.	1010 1110 0000 1011 0000 0000 0000 0100 _{two}
b.	1000 1101 0000 1000 0000 0000 0100 0000 _{two}

For the binary entries above, what instructions do they represent?

Review

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

I:	op	rs	rt	address / immediate
-----------	----	----	----	---------------------

J:	op	target address
-----------	----	----------------

op: basic operation of the instruction (opcode)

rs: first source operand register

rt: second source operand register

rd: destination operand register

shamt: shift amount

funct: selects the specific variant of the opcode (function code)

address: offset for load/store instructions ($\pm 2^{15}$)

immediate: constants for immediate instructions

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32 _{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34 _{ten}	n.a.
add immediate	I	8 _{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35 _{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43 _{ten}	reg	reg	n.a.	n.a.	n.a.	address

FIGURE 2.5 MIPS instruction encoding. In the table above, “reg” means a register number between 0 and 31, “address” means a 16-bit address, and “n.a.” (not applicable) means this field does not appear in this format. Note that add and sub instructions have the same value in the op field; the hardware uses the funct field to decide the variant of the operation: add (32) or subtract (34).

Name	Register number	Usage	Preserved on call?
\$zero	0	The constant value 0	n.a.
\$v0–\$v1	2–3	Values for results and expression evaluation	no
\$a0–\$a3	4–7	Arguments	no
\$t0–\$t7	8–15	Temporaries	no
\$s0–\$s7	16–23	Saved	yes
\$t8–\$t9	24–25	More temporaries	no
\$gp	28	Global pointer	yes
\$sp	29	Stack pointer	yes
\$fp	30	Frame pointer	yes
\$ra	31	Return address	yes

FIGURE 2.14 MIPS register conventions. Register 1, called \$at, is reserved for the assembler (see Section 2.12), and registers 26–27, called \$k0–\$k1, are reserved for the operating system. This information is also found in Column 2 of the MIPS Reference Data Card at the front of this book.

Problem 5.a

101011 10000 01011 00000000000000100

- opcode = $(101011)_2 = 43$
 - sw (I - type)
- rs = $(10000)_2 = 16$
 - \$s0
- rt = $(01011)_2 = 11$
 - \$t3
- address = $(000000000000000100)_2 = 4$
- a) `sw $t3, 4($s0)`

Problem 5.b

100011 01000 01000 0000000001000000

- opcode = $(100011)_2 = 35$
 - lw (I - type)
- rs = $(01000)_2 = 8$
 - \$t0
- rt = $(01000)_2 = 8$
 - \$t0
- address = $(0000000001000000)_2 = 64$
- **b)** `lw $t0, 64($t0)`

Problem 6

In the following problems, the data table contains the values for registers \$t0 and \$t1. You will be asked to perform several MIPS logical operations on these registers.

a.	\$t0 = 0x55555555, \$t1 = 0x12345678
b.	\$t0 = 0xBEADFEED, \$t1 = 0xDEADFADE

For the values in the table above, what is the value of \$t2 for the following sequence of instructions:

```
sll $t2, $t0, 4  
andi $t2, $t2, -1
```

Problem 6

		# \$t0 = 0101 0101 0101 0101 0101 0101 0101 0101
sll	\$t2, \$t0, 4	# \$t2 = 0101 0101 0101 0101 0101 0101 0101 0000
andi	\$t2, \$t2, -1	# \$t2 = 0101 0101 0101 0101 0101 0101 0101 0000 # -1 = 1111 1111 1111 1111 1111 1111 1111 1111 # \$t2 = 0101 0101 0101 0101 0101 0101 0101 0000

a) Final value of \$t2 is 0x55555550

b) Final value of \$t2 is 0xEADFEED0

Problem 7

For these problems, the table holds some logical operations that are not included in the MIPS instruction set. How can these instructions be implemented?

a.	<code>andn \$t1, \$t2, \$t3</code>	// bit-wise AND of \$t2, !\$t3
b.	<code>xnor \$t1, \$t2, \$t3</code>	// bit-wise exclusive-NOR

The logical instructions above are not included in the MIPS instruction set, but can be synthesized using one or more MIPS assembly instructions. Provide a minimal set of MIPS instructions that may be used in place of the instructions in the table above.

Problem 7

a) `andn $t1, $t2, $t3` # `and $t2, !$t3`

<code>nor \$t3, \$t3, \$zero</code>
<code>and \$t1, \$t2, \$t3</code>

b) `xnor $t1, $t2, $t3` # `exclusive-NOR`

<code>xor \$t1, \$t2, \$t3</code>
<code>nor \$t1, \$t1, \$zero</code>

Problem 8

For these problems, several instructions that are not included in the MIPS instruction set are shown.

a.	<code>abs \$t2, \$t3</code>	<code># R[rd] = R[rt] </code>
b.	<code>sgt \$t1, \$t2, \$t3</code>	<code># R[rd] = (R[rs] > R[rt]) ? 1:0</code>

For each instruction in the table above, find the shortest sequence of MIPS instructions that performs the same operation.

Problem 8

a) `abs $t2, $t3` `# R[rd] = | R[rt] |`

ABS:

`sub $t2, $zero, $t3` `# $t2 = - $t3`

`b1e $t3, $zero, done` `# if $t3 <= 0 then result is $t2`

`add $t2, $t3, $zero` `# if $t3 > 0 then result is $t3`

done:

Problem 8 (cont.)

b) `sgt $t1, $t2, $t3` `# R[rd] = (R[rs] > R[rt]) ? 1 : 0`

```
slt $t1, $t3, $t2
```

Problem 9.a

Consider the following MIPS loop:

```
LOOP: slt $t2, $0, $t1
      beq $t2, $0, DONE
      subi $t1, $t1, 1
      addi $s2, $s2, 2
      j LOOP
```

DONE:

- a) Assume that the register \$t1 is initialized to the value 10. What is the value in register \$s2 assuming \$s2 is initially zero?

Solution: 20

Problem 9.b

Consider the following MIPS loop:

```
LOOP:  slt $t2, $0, $t1
        beq $t2, $0, DONE
        subi $t1, $t1, 1
        addi $s2, $s2, 2
        j  LOOP
```

DONE:

- b) For each of the loops above, write the equivalent C code routine. Assume that the registers \$s1, \$s2, \$t1, and \$t2 are integers A, B, i, and temp, respectively.

Solution:

```
i = 10;
do {
    B += 2;
    i = i - 1;
} while ( i > 0)
```

Problem 9.c

Consider the following MIPS loop:

```
LOOP:  slt $t2, $0, $t1
        beq $t2, $0, DONE
        subi $t1, $t1, 1
        addi $s2, $s2, 2
        j  LOOP
```

DONE:

- c) For the loops written in MIPS assembly above, assume that the register \$t1 is initialized to the value N. How many MIPS instructions are executed?

Solution: $(5 * N) + 2$

Problem 10

For the following problems, the table holds C code functions. Assume that the first function listed in the table is called first. You will be asked to translate these C code routines into MIPS Assembly.

a.	<pre>int compare(int a, int b) { if (sub(a, b) >= 0) return 1; else return 0; } int sub (int a, int b) { return a-b; }</pre>
b.	<pre>int fib_iter(int a, int b, int n){ if(n == 0) return b; else return fib_iter(a+b, a, n-1); }</pre>


Implement the C code in the table in MIPS assembly. What is the total number of MIPS instructions needed to execute the function.

0044	compare:	addi \$sp, \$sp, -4	
0048		sw \$ra, 0(\$sp)	#we will call a proc. inside this proc., store \$ra
004C		add \$t0, \$a0, \$0	#store original arguments in temp registers
0050		add \$t1, \$a1, \$0	
0054		jal sub	
0058		addi \$t2, \$0, 1	#\$t2 = 1 (return 1)
005C		beq \$v0, \$0, exit	# if sub == 0 exit
0060		slt \$t3, \$0, \$v0	
0064		bne \$t3, \$0, exit	# if sub >0 exit
0068		add \$t2, \$0, \$0	# else \$t2 = 0 (return 0)
006C	exit:	add \$v0, \$t2, \$0	#\$v0 = \$t2 (return \$t2)
0070		lw \$ra, 0(\$sp)	
0074		addi \$sp, \$sp, 4	
0078		jr \$ra	
007C	sub:	sub \$v0, \$a0, \$a1	#return \$a0-\$a1
0080		jr \$ra	

```

int compare(int a, int b) {
    if (sub(a, b) >= 0)
        return 1;
    else
        return 0;
}
int sub (int a, int b) {
    return a-b;
}

```


 Instruction addresses