CS2100 Computer Organization
Tutorial 4
ANSWERS

---

1. Given below are the contents of some registers and memory locations, where mem(*A*) refers to the data word stored in the memory location at address *A*:

$s0 = 100	mem (100) = 1000
$s1 = 160	mem (160) = 1600
$s2 = 200	mem (200) = 2000
$s3 = 240	mem (240) = 2400
$s4 = 300	mem (300) = 3000
$t0 = 10000	mem (10000) = 100
$t1 = 15000	mem (15000) = 150
$t2 = 20000	mem (20000) = 200
$t3 = 25000	mem (25000) = 250
$t4 = 30000	mem (30000) = 300

For each of the MIPS instructions below, 'op' is an unknown opcode, which is not of our concern here. For each of the data operands in the instruction, if the format is legal, give its **memory address (if applicable)** and **the content inside**. Note that $s0, …, $s4, $t1, …, $t4 are symbolic register names, $zero is register 0 with content zero inside, and only the three addressing modes (register, immediate, and displacement) mentioned in class are considered as legal.

(a) op $t1, $t2
(b) op $s2, 100($zero)
(c) op $t4, 40($s2)
(d) op $s3, 200($zero)
(e) op $t3, $zero($t1)
(f) op $s1, 140($s1)

The answers for (a) have been done for you.

	Operand	**Target Memory Address**	**Content**
(a)	$t1 $t2	Not applicable Not applicable	15000 20000
(b)	$s2 100($zero)	Not applicable 100	200 1000
(c)	$t4 40($s2)	Not applicable 240	30000 2400
(d)	$s3 200($zero)	Not applicable 200	240 2000
(e)	$t3 $zero($t1)	Not applicable Illegal	25000 Illegal
(f)	$s1 140($s1)	Not applicable 300	160 3000

For question 2 below, we are exploring four different types of ISAs. MIPS belongs to the *Register-Register* style.

2. You are tasked to design a dedicated 16-bit processor to perform simple addition and would like to evaluate the design using the four different instruction set architecture (ISA) styles. For each of these architectures, the corresponding data movement and arithmetic operations are shown below. Note that the operands for the instructions are annotated with either @ for address, or $ for register.

ISA	Instructions	Explanation
Stack	**push** *@src* **pop** *@dest* **add**	Load value in *@src* onto top of stack. Transfer value at top of stack to *@dest.* Remove top two values in stack, add them, and load the sum onto top of stack.
Accumulator	**load** *@src* **add** *@src* **store** *@dest*	Load value in *@src* into accumulator. Add value in *@src* and value in accumulator, and put sum back into accumulator. Store the value in accumulator into *@dest.*
Memory-Memory	**add** *@dest*, *@src1*, *@src2*	Add values in *@src1* and *@src2*, and put the sum into *@dest*.
Register-Register	**load** $*reg*, *@src* **add** $*dest*, $*src1*, $*src2* **store** $*reg*, *@dest*	Load value in *@src* into *$reg*. Add values in *$src1* and *$src2*, and put sum into *$dest*. Store value in *$reg* into *@dest*.

Consider the following three C statements with integer variables a0, a1 and a2:

```
a0 = a1 + a2;
a1 = a0 + a2;
a2 = a0 + a1;
```

For each of the four architectural styles above, write the assembly code corresponding to the above C code. Assume that the values of a0, a1, a2 are pre-assigned to memory in addresses @a0, @a1, @a2 respectively. The registers are denoted by $r0 to $r4. First part of each code is given.

Stack	***Accumulator***	***Memory-Memory***	***Register-Register***
push @a1 **push** @a2 **add** **pop** @a0 **push** @a0 **push** @a2 **add** **pop** @a1 **push** @a0 **push** @a1 **add** **pop** @a2	**load** @a1 **add** @a2 **store** @a0 **add** @a2 **store** @a1 **add** @a0 **store** @a2	**add** @a0, @a1, @a2 **add** @a1, @a0, @a2 **add** @a2, @a0, @a1	**load** $r1, @a1 **load** $r2, @a2 **add** $r0, $r1, $r2 **store** $r0, @a0 **add** $r1, $r0, $r2 **store** $r1, @a1 **add** $r2, $r0, $r1 **store** $r2, @a2

3. This is a follow up on question 2. We studied the assembly code generated for four different storage architectures, namely stack, accumulator, memory-memory and register-register. For your reference, the code fragment corresponds to the following high level statements:

```
a0 = a1 + a2;
a1 = a0 + a2;
a2 = a0 + a1;
```

(a) Let us study the instruction encoding for this question. Assume that 3 bits will be used for the opcode, and minimal space will be used to represent 128 bytes of addressable memory. Moreover, for the Register-Register ISA, there are only 5 general-purpose registers available. Assume also a fixed-length instruction format, and that the memory is byte-addressable.

For each of the four architectures, what is the number of bits required for the longest instruction? Hence, what is the size, in number of bytes, of the instructions?

Partial answers are given below. Discuss.

	Number of bits for longest instruction	*Number of bytes*
Stack	10	2
Accumulator	10	2
Memory-Memory	**24**	4
Register-Register	**13**	**2**

- 3 bits needed to represent 5 registers;
- For a memory space of 128 bytes, 7 bits needed for a byte-addressable machine.
- A common question is why 4 bytes instead of 3 bytes for the 24-bit instruction? It could have been 3, but 4 bytes would permit a simpler design to avoid alignment problem (at the expense of more space). An instruction is usually fetched in a word, and a word usually contains 2, 4, or 8 bytes. It is uncommon to have a word that contains 3 bytes.

Stack	*Accumulator*	*Memory-Memory*	*Register-Register*
push @*src* **10** pop @*dest* **10** add **3**	load @*src* **10** add @*src* **10** store @*dest* **10**	add @*dest*, @*src1*, @*src2* **24**	load $*reg*, @*src* **13** add $*dest1*,$*src1*,$*src2* **12** store $*reg*, @*dest* **13**

The longest instruction for each of the four ISAs from left to right are 10, 10, 24 and 13 bits respectively.

To compute the instruction code size for a 16-bit processor, we consider the minimum number of 16-bit words necessary to encompass the bits. The sizes are therefore 2, 2, 4 and 2 bytes respectively. This information will be needed for the next question.

(b) What is the size (in number of bytes) of the code in question 2 for each of the four architectures? Which architecture is most efficient in terms of code size for this code?

Code size = number of instructions × size of each instruction

Stack: 12 × 2 = 24 bytes
Accumulator: 7 × 2 = 14 bytes
Memory-Memory: 3 × 4 = 12 bytes
Register-Register: 8 × 2 = 16 bytes

Hence Memory-Memory is the most efficient in terms of code size for this code.

4. [Past-year's exam question]
An ISA has 16-bit instructions and 5-bit addresses. There are two classes of instructions: class *A* instructions have one address, while class *B* instructions have two addresses. Both classes exist and the encoding space for opcode is completely utilized.

(a) What is the minimum total number of instructions?

(b) What is the maximum total number of instructions?

Answers

Class *A*: pppppppppppp xxxxx
Class *B*: qqqqqq xxxxx yyyyy
'pppppppppppp' and 'qqqqqq' are opcodes, 'xxxxx' and 'yyyyy' are addresses.

To obtain the minimum total number of instructions, we give class *B* ($2^6 - 1$) opcodes, leaving one 6-bit opcode as the prefix for class *A* 11-bit opcodes. Hence class *A* has 2^5 opcodes. Total = $(2^6 - 1) + (2^5) = 63 + 32 = 95$.

To obtain the maximum total number of instructions, we give class *B* only 1 opcode, leaving ($2^6 - 1$) prefixes for class *B* opcodes. Hence, class *B* has $(2^6 - 1) \times (2^5) = 63 \times 32 = 2016$ opcodes. Hence a total of 2017 instructions.