Lab2 & Ch2 review 1

Yuanqing Miao

Schedule

- **Lab2**
- **Chapter 2 review**

Lab2

Lab2

- Install MARS

https://dpetersanderson.github.io/download.html

Install Java

Java SE Development Kit 23.0.2 downloads

JDK 23 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions (NFTC).

JDK 23 will receive updates under these terms, until March 2025, when it will be superseded by JDK 24.

Linux **macOS** **Windows**

Product/file description	File size	Download
x64 Compressed Archive	228.77 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.zip (sha256)
x64 Installer	205.27 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.exe (sha256)
x64 MSI Installer	204.02 MB	https://download.oracle.com/java/23/latest/jdk-23_windows-x64_bin.msi (sha256)

Install Java

Setting the Environment Variables in Microsoft Windows

To set the `JAVA_HOME` and `PATH` environment variables in Microsoft Windows:

1. Click **Start**, **Control Panel**, **System**, and then **Advanced system settings**.
2. In the **System Properties** dialog box, on the Advanced tab, click **Environment Variables**.
3. Add the `JAVA_HOME` environment variable:
   a. In the **System Variables** section, click **New**.
   b. In the **Variable name** field, enter `JAVA_HOME`.
   c. In the **Variable value** field, enter the location where the JDK software is installed (for example, `C:\Program Files\Java\`*<java_version>*)
   d. Click **OK**.
4. Update the PATH environment variable to include the location of the Java executable files:
   a. In the **System Variables** section, select the PATH variable, and click **Edit**.
   b. In the **Variable value** field, insert `%JAVA_HOME%\bin;` in front of all the existing directories. Do not delete any existing entries; otherwise, some existing applications may not run.
   c. Click **OK**.
5. Exit the Control Panel.

Now we need to verify that you actually can use MARS successfully.

- When you run the previous program, what is printed?
- What is the value in register $t7 (in decimal) when the program ends?
- Set a breakpoint for the instruction at line 13 of the assembler source code. Run the program again; it should stop at the breakpoint. Now execute that one instruction. Which registers have changed as a result of executing that one instruction? You might need to continue past the breakpoint several times to see what's going on. Note that P&H COD Appendix A.10 has descriptions of all the instructions, but you can't just look up the answer. (You should look up the instructions in App. A.10, but the answer requires you to pull together several different pieces of information, not just one.)

You must mention all the registers that change not only for 1 iteration but for multiple. $t7 is not the only register that changes, keep running and you will see more changes. PC, lo, hi are registers that change per run so they are not very important but still worth mentioning in the lab report.

Put the answers to these three questions in your Lab. You have to upload the source codes used as well, all on one zip file.

Chapter 2 review 1

Q1

- For the following MIPS assembly instructions above, what is a corresponding C statement? f, g, h and i are assigned to registers \$s0, \$s1, \$s2 and \$s3, respectively.

```
add  f, g, h
add  f, i, f
```

Add and subtract, three operands

- Two sources and one destination

```
add a, b, c  # a gets b + c
```

```
f = g + h + i
```

Q2

- For the following C statement, what is the corresponding MIPS assembly code? Assume that the variables i, and j are assigned to registers $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.
- b = a << 2; a = 1111 -> 1100

```
B[8] = A[i - j];
```

```
sub $t0, $s3, $s4
sll     $t0, $t0, 2
add $t1, $s6, $t0
lw $t2, 0($t1)
sw $t2, 32($s7)
```

```
Address of A[i-j] : A + (i-j)*4
Address of B[8] : B + 8*4
```

Q3

- Assume that registers $s0 and $s1 hold the values 0x80000000 and 0xD0000000, respectively.
- a. What is the value of $t0 for the following assembly code?
  add $t0, $s0, $s1

```
  1000 0000 0000 0000 0000 0000 0000 0000  ($s0 = -2³¹)
+ 1101 0000 0000 0000 0000 0000 0000 0000  ($s1 = -0x30000000)
-----------------------------------------------------------
  0101 0000 0000 0000 0000 0000 0000 0000  ($t0 = 0x50000000)
```

In MIPS, overflow occurs only when adding two numbers of the same sign produces a result of the opposite sign.

Overflow

c. For the contents of registers $s0 and $s1 as specified above, what is the value of $t0 for the following assembly code?

`sub $t0, $s0, $s1`

The **sub** (`sub $t0, $s0, $s1`) instruction performs:

$$t0 = s0 - s1$$

no overflow

Since subtraction can be rewritten as **addition with the negation**:

$$t0 = s0 + (-s1)$$

```
  1000 0000 0000 0000 0000 0000 0000 0000  ($s0 = 0x80000000)
+ 0011 0000 0000 0000 0000 0000 0000 0000  (-$s1 = 0x30000000)
------------------------------------------------------------
  1011 0000 0000 0000 0000 0000 0000 0000  ($t0 = 0xB0000000)
```

Q4

- Provide the type and hexadecimal representation of following instruction:
  `sw $t1, 32($t2)`

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

Immediate arithmetic and load/store instructions

- rs: destination or source register number
- rt: destination or source register number
- Constant: -2^{15} to $+2^{15} - 1$
- Address: offset added to base address in rs

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- `sw $t1, 32($t2)`

Binary Breakdown:

- **Opcode** for `sw`: `101011` (6 bits)
- **rs** (`$t2` = `$10`): `01010` (5 bits)
- **rt** (`$t1` = `$9`): `01001` (5 bits)
- **Immediate** (`32` in decimal = `0000 0000 0010 0000` in 16-bit binary)

i-type, 0xAD490020

- 101011 01010 01001 0000000000100000
- A D 4 9 0 0 2 0

Q5

- Translate the following MIPS code to C. 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.

```
addi $t0, $s6, 4       # t0 = A + 4 (Address of A[1])
add $t1, $s6, $0       # t1 = A (Base address of A)
sw $t1, 0($t0)         # Store A at A[1]
lw $t0, 0($t0)         # Load A[1] into t0
add $s0, $t1, $t0      # f = A + A
```

Q6

- Find a sequence of MIPS instructions that extracts bits 16 down to 11 from register $t0 and uses the value of this field to replace bits 31 down to 26 in register $t1 without changing the other 26 bits of register $t1.
  - Bits 16-11 from $t0 are placed in bits 31-26 of $t1.
  - The other 26 bits of $t1 remain unchanged.

1. **Extract Bits 16-11 from `$t0`**
   - `srl $t0, $t0, 11` → Moves bits **16-11** to **5-0**.
2. **Move Extracted Bits to Position 31-26**
   - `sll $t0, $t0, 26` → Moves bits **5-0** to **31-26**.
3. **Create a Mask to Clear Bits 31-26 in `$t1`**
   - `ori $t2, $0, 0x03ff` → Loads `0x03FF` (10-bit value).
   - `sll $t2, $t2, 16` → Moves bits to position `0x03FF0000`.
   - `ori $t2, $t2, 0xffff` → Completes mask as `0x03FFFFFF`.
4. **Apply the Mask to `$t1`**
   - `and $t1, $t1, $t2` → Clears bits **31-26**, preserving others.
5. **Insert Modified Bits from `$t0` into `$t1`**
   - `or $t1, $t1, $t0` → Merges extracted bits into `$t1`.

Q7

- For the following C statement, write a minimal sequence of MIPS assembly instructions that does the identical operation. Assume $t1 = A, $t2 = B, and $s1 is the base address of C.

```
A = C[0] << 4;
```

```
lw $t3, 0($s1)

sll $t1, $t3, 4
```

Q8

- Assume $t0 holds the value 0x00101000 . What is the value of $t2 after the following instructions?

```
slt $t2, $0, $t0  # Set $t2 = 1 if $0 < $t0, else set $t2 = 0
bne $t2, $0, ELSE # Branch to ELSE if $t2 is NOT zero
j DONE            # Jump to DONE
ELSE: addi $t2, $t2, 2  # Increment $t2 by 2 (only executes if branch taken)
DONE:
```

$t2 = 3

Q9

- Write the MIPS assembly code that creates the 32-bit constant `0010 0000 0000 0001 0100 1001 0010 0100` (two) and stores that value to register `$t1`

```
Genererally, all solutions are similar:
lui $t1, top_16_bits
ori $t1, $t1, bottom_16_bits
```

```
lui $t1, 0x2001       # Load upper 16 bits (0010 0000 0000 0001) into $t1
ori $t1, $t1, 0x4924 # OR with lower 16 bits (0100 1001 0010 0100)
```