HW2: MIPS ISA

Due March 5th , 2023, 11:59pm

extra credit available for early submissions!

There are 2 parts to this assignment: part 1: written exercises for Ch2; part 2: MIPS programming. Basic rules regarding assignment implementation and submission:

- Total points of this assignment: 100.
- **Team Allowed. Maximum 2 per Team**. You must include names and G#'s of **BOTH** group members in **ALL** submitted files**.**

Submission Instructions

- Submit to **gradescope**. A link to gradescope will be available from Blackboard. There will be two assignments on gradescope, one for part 1 and one for part 2. You will lose "off the top" points if you fail to follow the submission instructions.
- For Part1, follow the submission instruction from gradescope. You will need to upload a **PDF** and **specify** on which page we can find your answer for each question. Make sure your answer is legible.
  - **Up to -5 points** for incorrect format or not specifying pages.
- For Part 2, name your MIPS source code (`cs465_hw2_subroutines.asm` only)

  `cs465_hw2_username1[_userame2].asm`
  - Here username is the first part of your GMU email address. Make sure to include both members' names if you have two people in the team.
  - For example: cs465_hw2_yzhong.asm
  - Do **NOT** submit it as a .pdf.
- For teams with two people:
  - For each part, only **one** team member should submit to Gradescope.
  - You won't be able to specify the team when you submit. Instead, you must **include names and G#'s of both group members** in all submitted files**.** We will use the provided group information to assign the same grade to both members.
- **Plagiarism is not permitted in any form. I enforce the university honor code.**

Part 1. Written Exercise for MIPS ISA (50%)

Notes:

- **A large portion of (or all) points will be taken off if you do not include detailed calculation in your answer.** You must show steps to justify your answer.
- **Answers must be legible**, especially if you scan to generate your submission.

1. (10 pts) MIPS encoding. Encode each of the following MIPS instructions into a 32-bit machine code. You must first specify the decimal value of each field of the machine code, then give the complete machine code as a hexadecimal value.

Example: `add $t0, $s1, $s2`

Encoded as:

opcode	rs	rt	rd	shamt	func
0	17	18	8	0	32

Machine code: 0x02324020

```
1.1   sub $s0, $t0, $v0
1.2   lw $t8, 2023($s2)
```

2. (21pts) MIPS decoding.
2.1. Fill in the table to provide the details of each machine code.

Machine Code	Opcode (in assembly)	Format Type (I/J/R)	Source Register(s)	Destination Register
0xAD0A 001A				
0x1509 2023				
0x0800 252A				

Note:

- Only answer with general purpose register(s), i.e., no need to specify the usage of **`$PC`**.
- If an instruction does not read from any register or does not update any register, use N/A as the answer to source or destination register.

2.2. For each machine code from **2.1**, assume they are at address **0x0010 0FFC**. How would each of the instruction update **`$PC`**? List the updated value of **`$PC`** for each: for conditional branches, list two possible **`$PC`** values. **Show** address calculation steps to get full credit.

3. (4 points) Consider the following MIPS code:

```
LOOP: slt $t2, $0, $t1
      beq $t2, $0, DONE
      addi $t1, $t1, -3
      addi $s2, $s2, 1
      j LOOP
DONE:
```

Assuming that $t1 = 40 and $s2 = 0 initially.
1) What is the final value of $s2? You need to show your calculation.
2) How many MIPS instructions will be executed for the given sequence? You need to show your calculation to get full credit.

4. (15 points) Translate function `foo` into MIPS assembly language. <u>You must follow MIPS register usage conventions for function calls/returns as discussed in class</u>. Check the Appendix of this document for a quick reference. Assume the function declaration of **`bar`** is "**`int bar(int x);`**" and a label **`bar`** marks the starting point of its function body. You can assume that function **`bar`** follows the same set of MIPS conventions. No need to implement **`bar`**.

- **You can NOT use any pseudo-instructions.**
- **You need to comment you code to get full credit for this question.**

```
void foo(int *vals) {
    while (bar(*vals) > 0) {
        vals++;
    }
}
```

Part 2. MIPS Programming (50%)

MIPS Decoder. For this assignment, you will write a program to accept a 32-bit machine code (as a hexadecimal string) from the user, decode, and report certain features of the decoded MIPS instruction.

cs465_hw2_decoder.asm: This is the main program **provided to you** which performs the following tasks:

- Accept a machine instruction word in hexadecimal as one string from the user.
- Call subroutine **`atoi`**(details below) to extract the numeric value from the input string.
- Call subroutine **`get_insn_code`**(details below) and print an instruction code of the instruction. If the instruction is not supported, print **`"invalid"`**. Check the Appendix for the insn_code we define for each supported MIPS instruction.
- Call subroutine **`get_src_regs`**(details below) and print the register number(s) returned. If the instruction is not supported, print **`"invalid"`**. If there is no source register (i.e., the instruction does not read from any general-purpose register), print **`"N/A"`**.
- Call subroutine **`get_next_pc`**(details below) and print the address(es) returned. If the instruction is not supported, print **`"invalid"`**.
- Provided subroutine **`void step(int step)`**
    - This subroutine prints a message to report that the specified **`step`** is done.

cs465_hw2_subroutines.asm: This is the file that **you need to complete** and implement the required subroutines for the main program to call. We provide a template for you to start.

- **For all required subroutines**: at the end of the subroutine you must call **`step()`** with the specified step number.
- **`uint32_t atoi(char* start_addr)`**
    - Step number: **`1`**
    - Argument: **`start_addr`** is the starting address of a string buffer. The buffer stores a hexadecimal string like "**`0000001A`**".
    - Return: an unsigned 32-bit integer calculated from the given 8-digit hexadecimal string.
    - Assumptions:
        - The string buffer has a string of 8 characters.
        - Only capital case letters (**`'A'`** to **`'F'`**) and decimal digits will be used for the input.
        - All input are valid hexadecimal values -- no need to check and report invalid digits for this homework.

- **`uint32_t get_insn_code(uint32_t instruction)`**
    - Step number: **`2`**
    - Argument: **`instruction`** is a 32-bit unsigned integer representing a MIPS machine word.
    - Return: an unsigned 32-bit integer as the unique "*instruction code*" of the instruction. Check the Appendix for the unique number we define for each supported instruction.
        - Return 0xFFFFFFFF for invalid instructions.
    - Assumption: You only need to support a subset of MIPS instructions: **`sub, addi, slt, lw, sw, bne, j, jal`**. All other instructions are invalid. Check the Appendix for details.

- **`uint32_t get_src_regs(uint32_t instruction)`**
    - Stage number: **`3`**
    - Argument: **`instruction`** is a 32-bit unsigned number representing a MIPS machine word.

- Return: one or two integers representing the source register(s) of this instruction. A source register is a general-purpose register that this instruction reads from to get a source operand.
    - A normal return should be within the range of [0,31].
    - If the instruction only has one source register, return its number in $v0 and set $v1 to be 32.
    - If the instruction has two source registers, return $rs in $v0 and $rt in $v1.
    - Set $v0 to 32 if the instruction has no source registers.
    - Set $v0 to 0xFFFFFFFF for invalid instructions.

- **`uint32_t get_next_pc(uint32_t instruction, uint32_t addr)`**
    - Step number: **`4`**
    - Arguments: **`instruction`** is a 32-bit unsigned number representing a MIPS machine word; **`addr`** is a 32-bit unsigned number representing the byte address of the given **`instruction`**.
        - **Note**: the 2nd argument (**`addr`**) is fixed to be zero when **`get_next_pc`** is called in the provided **`cs465_hw2_decoder.asm`**. But we will change this and use varied word addresses in testing. Do not assume the address (i.e. current $PC) is always zero in your implementation.
    - Return: one or two integers as the byte addresses of the next instruction(s) in control flow.
        - For sequential and jump instructions, return one unsigned 32-bit integer representing the address of the next instruction we will fetch and execute in $v0 and set $v1 to be 0xFFFFFFFF.
        - For conditional branch instruction, return two integers representing the addresses of the next instruction we will fetch and execute if branch is taken (in $v1) and if branch is not taken (in $v0).
        - Set $v0 to be 0xFFFFFFFF for invalid instructions. You can assume there is no address overflow nor other restrictions for the $PC updating.

Coding Requirements:

- Do NOT change the provided **`cs465_hw2_decoder.asm`**. You should **only** change and submit **`cs465_hw2_subroutines.asm`**. Check the submission instructions to **rename your file** when you are ready to turn in the homework.
- We have marked clearly the start and end of all required subroutines in the provided template file **`cs465_hw2_subroutines.asm`**. All required subroutines are declared global. Do NOT change these. For example, these two lines declare your subroutine **`atoi()`**

```
.globl atoi
atoi:
```

- **Your subroutines must follow MIPS register usage conventions as discussed in our textbook and slides.** See Appendix of this document for a quick reference.
- You must call the provided **`step()`** before returning from each required subroutine. You will not get credit if you fail to do so even if the output matches the required one.
- You code must be very well commented. A description of the algorithm you use must be included in your comments.

Hints:

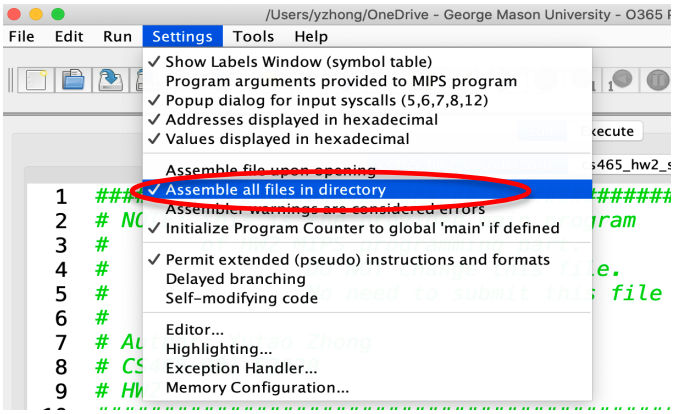
- You can use your own MIPS code from HW1.
- Additional helper functions can be useful.
- ASCII encoding: http://en.wikipedia.org/wiki/ASCII#Printable_characters .
- System calls: http://courses.missouristate.edu/KenVollmar/mars/Help/SyscallHelp.html
- MIPS encoding: https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_help.html

How to run:

- **From a command line:**
  - Include both files in one command and make sure to follow this order:

```
java -jar mars.jar cs465_hw2_decoder.asm cs465_hw2_subroutines.asm
```

- **From MARS:**
  - Place both .asm files in the same folder.
  - From **Settings**, select "**Assemble applies to all files in directory**".
  - You can open both .asm files for editing but always make sure *cs465_hw2_decoder.asm* is the currently opened file when you assemble/run.



Grading rubric:

• Code assembled with no error	5/50
• Code well commented	5/50
• Correct decoding in a sequence of steps	30/50
• MIPS call/return conventions followed	10/50

- We will be comparing the generated output file and the expected output file for grading.
- **No late work allowed after 24 hours.**
  - Late submission automatically uses one of your tokens.
- **Extra credit for early submissions:**
  - 1% extra credit rewarded for every 24 hours your submission made before the due time.
  - Up to 3% extra credit will be rewarded.
  - Your latest submission will be used for grading and extra credit checking. You CANNOT choose which one counts.
- You will need to make two separate submissions for Part1 and Part2 respectively. The later one of those two will be used to determine whether your HW2 is early / normal/ late.

Appendix: MIPS ISA

There are MIPS conventions you need to follow and a subset of MIPS instructions that you need to support in order to implement the coding part of this assignment. You can check the details from our textbook/lecture slides. This section serves as a quick reference for you.

MIPS instructions that you must support:

You are required to support only a subset of MIPS instructions. Use the table below as your reference. For all other inputs, your program should return/print "invalid".

MIPS	insn_code	opcode	funct	MIPS	insn_code	opcode	funct
sub	0x0	0x00	0x22	sw	0x4	0x2b	-
addi	0x1	0x08	-	bne	0x5	0x05	-
slt	0x2	0x00	0x2a	j	0x6	0x02	-
lw	0x3	0x23	-	jal	0x7	0x03	-

MIPS conventions you must follow:

You are required to follow MIPS conventions for register usages and function calls/returns. Check the textbook and lecture slides for details. Here is a quick summary:

- Function calls/returns must use jal and jr.
- Arguments and return values must use $a and $v registers in order. That is, the first argument must use $a0, the second argument must use $a1, etc. If there is only one 32-bit return value, $v0 must be used.
- Use stack and $sp for local data and register spilling.
- Divide the maintenance of shared registers between the caller and callee following the table below:

Reg Name	Reg Number	Usage	Whose Responsibility?	
			Caller	Callee
$v0-$v1	2-3	Return value	√	
$a0-$a3	4-7	Argument	√	
$t0-$t7, $t8-$t9	8-15,24-25	Temporaries	√	
$s0-$s7	16-23	Saved		√
$sp	29	Stack pointer		√
$ra	31	Return address	√ *	

* The value of $ra will be changed by jal.

Sample runs (user input in underlined blue, with newline displayed explicitly):

Note: Below are multiple sample runs obtained from command-line executions in a prompt. Newline and user input display will be different if you run directly in MARS but all numbers/strings should match.

```
Enter a MIPS machine word in hex: 0x016B4822↵
========================
Step Completed: 1
Input: 0x016b4822
========================
Step Completed: 2
Instruction Code: 0
========================
Step Completed: 3
Source Register(s): 11, 11
========================
Step Completed: 4
Next PC(s): 0x00000004
```

```
#sub $t1, $t3, $t3
#assume current PC is 0x0
```

```
Enter a MIPS machine word in hex: 0x23302023↵
========================
Step Completed: 1
Input: 0x23302023
========================
Step Completed: 2
Instruction Code: 1
========================
Step Completed: 3
Source Register(s): 25
========================
Step Completed: 4
Next PC(s): 0x00000004
```

```
#addi $s0, $t9, 0x2023
#assume current PC is 0x0
```

```
Enter a MIPS machine word in hex: 0x16A41111↵
========================
Step Completed: 1
Input: 0x16a41111
========================
Step Completed: 2
Instruction Code: 5
========================
Step Completed: 3
Source Register(s): 21, 4
========================
Step Completed: 4
Next PC(s): 0x00000004, 0x00004448
```

```
#bne $s5,$a0,0x1111
#assume current PC is 0x0
```

```
Enter a MIPS machine word in hex: 0x0C011110↵
========================
Step Completed: 1
Input: 0x0c011110
========================
Step Completed: 2
Instruction Code: 7
========================
Step Completed: 3
Source Register(s): N/A
========================
Step Completed: 4
Next PC(s): 0x00044440
```

```
#jal 0x01110
#assume current PC is 0x0
```

```
Enter a MIPS machine word in hex: 0xA1234567↵
========================
Step Completed: 1
Input: 0xa1234567
========================
Step Completed: 2
Instruction Code: Invalid
========================
Step Completed: 3
Source Register(s): Invalid
========================
Step Completed: 4
Next PC(s): Invalid
```

```
#not supported
```