HW2: MIPS ISA

Due October 11, 2020, 11:55pm

If you have questions, use the piazza forums (and professor/TA office hours) to obtain assistance.

Task:

There are 2 parts to this assignment: part 1: written exercises for Ch1; part 2: MIPS programming. Basic rules regarding assignment implementation and submission:

- Total points of this assignment: 100.
- **Team Allowed. Maximum 2 per Team**. You should be able to specify your team on gradescope. To be safe, include **names and G#'s of group members in ALL submitted files.**

Submission Instructions

- Submit to **gradescope**. A link to gradescope is available from Blackboard. There will be two assignments on gradescope, one for part 1 and one for part 2.
- For Part1, follow the submission instruction from gradescope. You will need to upload a PDF and specify which page we can find your answer. Make sure your answer is legible.
- For Part 2, name your MIPS source code as **`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.
- **Plagiarism is not permitted in any form. I enforce the university honor code.**

Part 1. Written Exercise for Performance (45%)

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   srl $t3, $t4, 10
1.2   lw $t0, 32($s1)
```

2. (15pts) MIPS decoding. Assume the given MIPS instruction is at address **0x0080 0100**. Answer the following questions for the given instruction word:

- What is the opcode (in assembly) of this instruction?

- What type (I/J/R) is this instruction?
- Which registers are read / updated in executing this instruction? You can use either assembly names or register numbers in your answer.
- What is the address of the next instruction we will fetch and execution? If the instruction is a conditional branch, give two possible addresses. Give the answer as a hexadecimal value and show the address calculation steps to get full credit.

2.1 **0xAE09 0020**
2.2 **0x1253 0011**
2.3 **0x0800 AABB**

3. (6 pts) Consider the following MIPS code:

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

Assuming that $t1 = 20 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.

4. (14 points) Translate function foo into MIPS assembly language. You must follow MIPS register usage conventions for function calls/returns as discussed in class. 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 MPS conventions but do NOT need to implement it. **You can NOT use 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(55%)

MIPS Decoder. For this assignment, you will write a program to accept machine code (as hexadecimal strings) 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_type`(details below) and print the format type (I/J/R) of the instruction. If the instruction is not supported, print `"invalid"`.
- Call subroutine `get_dest_reg`(details below) and print the number of register that gets updated by the instruction as a decimal value. If there is no destination register, print `"N/A"`. If the instruction is not supported, print `"invalid"`.
- Call subroutine `get_next_pc`(details below) and print the address(es) of the next instruction(s) in control flow. If the instruction is not supported, print `"invalid"`.
- Provided subroutine `void report(int stage)`
  - This subroutine prints a message to report that the specified `stage` 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 `report()` with the specified stage number.
- **`unsigned int atoi(char* start_addr)`**
  - Stage number: `1`
  - Argument: `start_addr` is the starting address of a string buffer. The buffer stores a hexadecimal string like "`0000001A`". You can assume the string always has 8 valid hexadecimal characters.
  - Return: an unsigned integer calculated from the given 8-digit hexadecimal string.
  - Assumptions:
    - The input machine word from the user is a string of 8 characters.
    - You can assume only capital case letters (`'A'` to `'F'`) will be used for the input.
    - You can assume all input are valid hexadecimal values, no need to check and report invalid digit for this homework.

- **`int get_type(unsigned int instruction)`**
  - Stage number: `2`
  - Argument: `instruction` is a 32-bit unsigned number representing a MIPS machine word.
  - Return: an integer with the following possible values:
    - 0 for R type, 1 for I type, 2 for J type, -1 for invalid instructions
  - Assumption: You can assume only a subset of MIPS instructions are supported: `add, addi, and, slt, lw, sw, beq, j`. Check the Appendix for details. All other instructions are invalid.

- **`int get_dest_reg(unsigned int instruction)`**
  - Stage number: `3`
  - Argument: `instruction` is a 32-bit unsigned number representing a MIPS machine word.
  - Return: an integer representing the register number that will be updated by executing of this instruction. Note: some instruction might update $rt instead of $rd.
    - A valid return should be within the range of [0,31]. No need to throw a warning about updating register zero for this task.
    - Return 32 if no register gets updated by the instruction. Return -1 for invalid instructions.

- **`int get_next_pc(unsigned int instruction, unsigned int addr)`**
  - Stage number: `4`
  - Argument: `instruction` is a 32-bit unsigned number representing a MIPS machine word; `addr` is a 32-bit unsigned number representing the address (PC) of the given `instruction`.

- Return: one or two integers as the addresses of the next instruction(s) in control flow.
    - You can assume the next pc is never going to be -1.
    - For sequential and jump instructions, return an integer representing the address of the next instruction we will fetch and execute in $v0 and set $v1 to be -1.
    - 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 the next instruction if branch is not taken (in $v0).
    - Set -1 for $v0 for invalid instructions. You do not need to check for addr overflow.

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 `report()` before return 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.

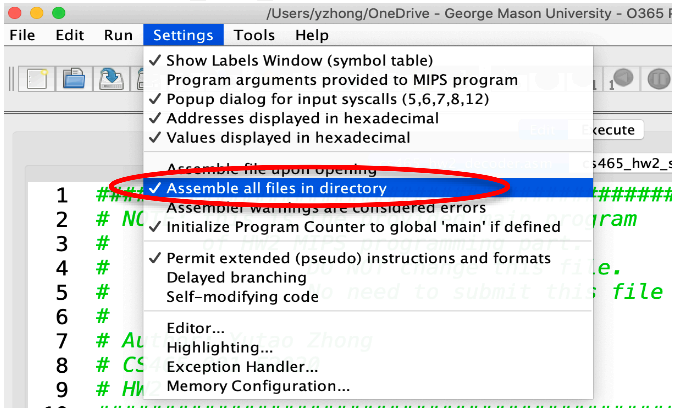
How to run:

- **From a command line:**

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

- **From IDE:**

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



✓ Show Labels Window (symbol table)
Program arguments provided to MIPS program
✓ Popup dialog for input syscalls (5,6,7,8,12)
✓ Addresses displayed in hexadecimal
✓ Values displayed in hexadecimal
Assemble file upon opening
✓ Assemble all files in directory
Assembler warnings are considered errors
✓ Initialize Program Counter to global 'main' if defined
✓ Permit extended (pseudo) instructions and formats
Delayed branching
Self-modifying code

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: http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html

Grading rubric:

• Code assembled with no error	5/55
• Code well commented	5/55
• Required subroutines	30/55
• Stages reported as required	5/55
• MIPS call/return conventions followed	10/55

Appendix: MIPS ISA and MARS

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	opcode	funct	Type	MIPS	opcode	funct	Type
add	0x00	0x20	R (0)	addi	0x08	X	I (1)
and	0x00	0x24	R (0)	slt	0x00	0x2a	R (0)
lw	0x23	X	I (1)	sw	0x2b	X	I (1)
beq	0x04	X	I (1)	j	0x02	X	J (2)

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: 0x012A4020↵
========================
Stage passed: 1
Input: 0x012a4020
========================
Stage passed: 2
Instruction Type: 0
========================
Stage passed: 3
Destination Register: 8
========================
Stage passed: 4
Next PC(s): 0x00000004
```

```
#add $t0, $t1, $t2
#current PC: 0x0
```

```
Enter a MIPS machine word in hex: 0xAE1F0010↵
========================
Stage passed: 1
Input: 0xae1f0010
========================
Stage passed: 2
Instruction Type: 1
========================
Stage passed: 3
Destination Register: N/A
========================
Stage passed: 4
Next PC(s): 0x00000004
```

```
#sw $ra, 0x10($s0)
#current PC: 0x0
```

```
Enter a MIPS machine word in hex: 0x08200035↵
========================
Stage passed: 1
Input: 0x08200035
========================
Stage passed: 2
Instruction Type: 2
========================
Stage passed: 3
Destination Register: N/A
========================
Stage passed: 4
Next PC(s): 0x008000d4
```

```
#j 0x200035
#current PC: 0x0
```

```
Enter a MIPS machine word in hex: 0x11090110↲
========================
Stage passed: 1
Input: 0x11090110
========================
Stage passed: 2
Instruction Type: 1
========================
Stage passed: 3
Destination Register: N/A
========================
Stage passed: 4
Next PC(s): 0x00000004, 0x00000444
```

```
#beq t0, t1, 0x0110
#current PC: 0x0
```

```
Enter a MIPS machine word in hex: 0x70000000↲
========================
Stage passed: 1
Input: 0x70000000
========================
Stage passed: 2
Instruction Code: invalid
========================
Stage passed: 3
Destination Register: invalid
========================
Stage passed: 4
Source Register(s): invalid
```

```
#not supported
```