Project: MIPS Simulator

Write a MIPS architecture simulator program in C/C++.

Input:

- MIPS program in binary code
- Number of cycles to run
- Initial register state (**optional**)

Output:

Print the state of the processor after each cycle. Process state includes the following information.

- PC
- Instruction
- Register contents
- Memory I/O

Example.

$ ./MIPS_Simulator {Program_binary} {Number_of_cycles} [Initial_register_state]

Cycle 1
PC: 0000
Instruction: 23BDFFF8
Registers:
[0] 00000000
[1] 00000101
[2] 00000010
…
[31] 00000000
Memory I/O:

Cycle 2
PC: 0004
Instruction: AFB00004
Registers:
[0] 00000000
[1] 00000101
[2] 00000010
…
[31] 00000000
Memory I/O: W 4 0000 1234FF10

Cycle 3
PC: 0008
Instruction: AFB00004
Registers:
[0] 00000000
[1] 00000101
[2] 00000010
…
[31] 00000000
Memory I/O: R 2 0000 FF10

Memory I/O: [RW] {bytes} {addr} {data}

At cycle 2, the processor writes 4-byte data “0x1234FF10” at address 0x0000.
At cycle 3, the processor reads 2 bytes from address 0x0000, which returns “0xFF10”.

{bytes} can be either 1, 2, or 4.
{addr} is 32-bit (but please print 16-bit).
{data} is {bytes}-bytes (and please print {bytes}-bytes.).

Phase 1.

Phase 1 is a drastically simplified version of the simulator, which gives PC and instruction information only. Also, assume that branch is **always taken** for this version.
This means that you don’t need to simulate all the behavior of instructions. You just need to properly decode instructions.

Input:

- MIPS program in binary code
- Number of cycles to run

Output:
Print the state of the processor after each cycle. Process state includes the following information.

- PC
- Instruction

Input file format
Input files are text files.

- Initial register state

This file consists of 32 lines, each line showing the register content of a register in hexadecimal, starting from R0 all the way to R31.

eg.

```
00000000
0000000F
000000FF
…
00000000
```

R0 = 0
R1 = 15
R2 = 255

- Compiled MIPS code file

This file contains one instruction per line, represented as a hexadecimal number.

eg.

```
23BDFFF8
AFB00004
AFBF0000
14800002
20A20001
08100012
14A00004
2084FFFF
20050001
0C100000
08100012
00808020
20A5FFFF
0C100000
2204FFFF
00402820
0C100000
08100012
8FB00004
8FBF0000
23BD0008
03E00008
```

The first line means: addi $sp, $sp, -8.

The second line means: sw $s0, 4($sp).

Notes

1. This simulator is not for a 64-bit processor. Stick to the textbook definition of the MIPS instruction set.
2. For assembler, you may use this online MIPS assembler: https://alanhogan.com/asu/assembler.php
3. For Phase 2, pipelining must be modeled exactly. However, pipeline hazard detection and forwarding is optional. Thus, your simulator will produce incorrect result if the program contains some pipeline hazard patterns, which is to be expected.
4. All the numbers in the input/output are hexadecimal, except for cycle count and register index. Do not prepend them with 0x.

Instructions to be supported:

- All the instructions in the Chapter 2 of textbook (Figure 2.1),
- Except LL (load linked word) and SC (store condition. word),

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set on less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	slti $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	sltiu $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

Submission Instruction for Phase 1

- Your source code that can compile on a uni0x machine without error.
    - There will be a penalty for compile error.
- Output text files for test examples. One output text file per example.
    - Do not submit screen capture images.
    - Test examples will be provided later.