UIC – ECE 366: Computer Organization - Spring 2020

Project 1: MIPS programming with MARS

In this project, you will write a MIPS assembly program and verify that it runs correctly with the simulator MARS. This program is a simple version of the "Perceptron Algorithm" widely used in AI, and will be used for subsequent projects of this class.

Simplified Perceptron Algorithm

The job is to "guess" the hidden parameters (weights) of A and B, in a simple linear function F(x, y) = Ax + By, where all the numbers are assumed to be integers. A and B are not given, but whether the result F(x[i], y[i]) is positive (>=0) or negative (<0), is given for a large number of sample points (x[1], y[1]), (x[2], y[2]), … The algorithm goes as follows:

```
Use a "guess" version of f(x, y) = ax + by, initialized with
a = b = 1, to go through all the sample points (x[i], y[i]), and
update (improve) a and b accordingly:

If f(x[i], y[i]) < 0 but F(x[i], y[i]) >= 0:
     update a = a + x[i];
     update b = b + y[i];

If f(x[i], y[i]) >= 0 but F(x[i], y[i]) < 0:
     update a = a - x[i];
     update b = b - y[i];

Otherwise (f and F have the same sign for x[i], y[i]):
   keep a, b, as they are.
```

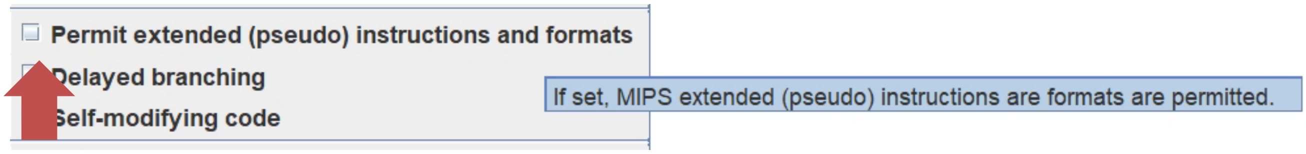
Restrictions:

You are limited to use the instructions listed in the MIPS reference card (attached in Appendix), NOT including the pseudo-instructions.

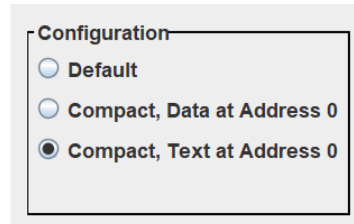
MARS Setup:

when programming your MIPS assembly code, MARS needs to be set with the following:

- "Permit extended (pseudo) instructions and formats" option UNCHECKED such that extended (pseudo) instructions are NOT permitted.



- MIPS Memory Configuration should be set to the 3rd option: Compact, Text at Address 0



(60%) Part A: generate the sample points via a "right-up" spiral.

Write MIPS code to generate n sample points from (x[1], y[1]) to (x[n], y[n]) in the following way: starting from (0, 0), always follow the sequence of going right, up, left, and down, in a "spiral" fashion that can reach any point exactly once, within enough steps.

Specifically:

first	1 step	right	(x1, y1) = (1, 0)		
then	1 step	up	(x2, y2) = (1, 1)		
then	2 steps	left	(x3, y3) = (0, 1)	(x4, y4) = (-1, 1)	
then	2 steps	down	(x5, y5) = (-1, 0)	(x6, y6) = (-1, -1)	
then	3 steps	right	(x7, y7) = (0, -1)	(x8, y8) = (1, -1)	(x9, y9) = (2, -1)
then	3 steps	up	(x10, y10) = (2, 0)	(x11, y11) = (2, 1)	(x12, y12) = (2, 2)
then	4 steps	left	…		
then	4 steps	down	…		

Other Specifics:

In the end, your program should store the word arrays:

```
x[1] at M[0x2020], followed by x[2] at 0x2024, x[3] at 0x2028, etc
y[1] at M[0x2420], followed by y[2] at 0x2424, y[3] at 0x2428, etc
```

Your program must begin with the following line of code which is used to initialize the number of sample points (n=100). Therefore, if we modify this instruction, we can change the number of points to generate. (This should work for Part A, B, and C)

```
addi $3, $0, 100     # n = 100
```

(80%) Part B: evaluate the classification array for F(x, y)

Your program must at the beginning use the following 2 lines of code to specify A and B:

```
addi $1, $0, 5       # A = 5
addi $2, $0, -6      # B = -6
```

Again, if we modify these instructions, we can change A, B to other values.

Your program should use these given `A, B` values, to provide the "classification" results:

```
C[i] = 1     when F(x, y) = A * x[i] + B * y[i] > = 0
C[i] = -1    when F(x, y) = A * x[i] + B * y[i] < 0
```

In the end, your program should store the word array:

`C[1] at M[0x2820], followed by c[2] at 0x2824, c[3] at 0x2828, etc.`

(100%) Part C: perceptron algorithm

Begin at a = b = 1, use the arrays x, y, and C in memory, to update a and b, according to the perceptron algorithm.

In the end, write the final a and b values in memory:

```
a at M[0x2000]
b at M[0x2004]
```

(10% extra credit) NO multiplication instructions

Your implementation should NOT include any multiplication instructions, but still achieve reasonable runtime.

Report components:

1. General intro and Q&A (-2 points each if answer is missing)

    a) Which parts (A, B, C, extra credit) does your program achieve?
    b) How many hours have you spend in total on this project?
    c) What was the most difficult (or time-consuming) part of this project?

2. Program functionality

    Run your program for each of the following configurations:

    ```
    a) A = 5,    B = -6,   n = 100
    b) A = 0,    B = 5,    n = 50
    c) A = 7,    B = 0,    n = 150
    d) A = 2,    B = 13,   n = 200
    ```

    Provide MARS screenshots of your program's results for each of the above configurations, including:

    - final results for a, b, x, y, c
        - `show MARS' data memory content at relevant regions of:`
            - `0x2000, 0x2020, 0x2420, 0x2820.`
    - instruction count of completing your program
        - `use MARS Tools -> Instruction Statistics`

    We will also test your program with other values.

3. Appendix

    Include your MIPS assembly code here (-5 if missing).

Deadline: **week 4 Sunday (end of the day) on Bb.**

What to submit: upload the following **`two files`** on Bb

- **`report.pdf`** the report specified above.
- **`p1.asm`** your MIPS assembly code (so we can run your code on MARS – missing it will result in -5 points).

Appendix: allowed MIPS instructions – choose up to 15 different ones among them

add	rd, rs, rt	Add	rd = rs + rt	R 0 / 20
sub	rd, rs, rt	Subtract	rd = rs − rt	R 0 / 22
addi	rt, rs, imm	Add Imm.	rt = rs + $imm_{\pm}$	I 8
addu	rd, rs, rt	Add Unsigned	rd = rs + rt	R 0 / 21
subu	rd, rs, rt	Subtract Unsigned	rd = rs − rt	R 0 / 23
addiu	rt, rs, imm	Add Imm. Unsigned	rt = rs + $imm_{\pm}$	I 9
mult	rs, rt	Multiply	{hi, lo} = rs * rt	R 0 / 18
div	rs, rt	Divide	lo = rs / rt; hi = rs % rt	R 0 / 1a
multu	rs, rt	Multiply Unsigned	{hi, lo} = rs * rt	R 0 / 19
divu	rs, rt	Divide Unsigned	lo = rs / rt; hi = rs % rt	R 0 / 1b
mfhi	rd	Move From Hi	rd = hi	R 0 / 10
mflo	rd	Move From Lo	rd = lo	R 0 / 12
and	rd, rs, rt	And	rd = rs & rt	R 0 / 24
or	rd, rs, rt	Or	rd = rs | rt	R 0 / 25
nor	rd, rs, rt	Nor	rd = ~(rs | rt)	R 0 / 27
xor	rd, rs, rt	eXclusive Or	rd = rs ^ rt	R 0 / 26
andi	rt, rs, imm	And Imm.	rt = rs & imm_{0}	I c
ori	rt, rs, imm	Or Imm.	rt = rs | imm_{0}	I d
xori	rt, rs, imm	eXclusive Or Imm.	rt = rs ^ imm_{0}	I e
sll	rd, rt, sh	Shift Left Logical	rd = rt << sh	R 0 / 0
srl	rd, rt, sh	Shift Right Logical	rd = rt >>> sh	R 0 / 2
sra	rd, rt, sh	Shift Right Arithmetic	rd = rt >> sh	R 0 / 3
sllv	rd, rt, rs	Shift Left Logical Variable	rd = rt << rs	R 0 / 4
srlv	rd, rt, rs	Shift Right Logical Variable	rd = rt >>> rs	R 0 / 6
srav	rd, rt, rs	Shift Right Arithmetic Variable	rd = rt >> rs	R 0 / 7
slt	rd, rs, rt	Set if Less Than	rd = rs < rt ? 1 : 0	R 0 / 2a
sltu	rd, rs, rt	Set if Less Than Unsigned	rd = rs < rt ? 1 : 0	R 0 / 2b
slti	rt, rs, imm	Set if Less Than Imm.	rt = rs < $imm_{\pm}$? 1 : 0	I a
sltiu	rt, rs, imm	Set if Less Than Imm. Unsigned	rt = rs < $imm_{\pm}$? 1 : 0	I b
j	addr	Jump	PC = PC&0xF0000000 | ($addr_{0}$<< 2)	J 2
jal	addr	Jump And Link	$ra = PC + 8; PC = PC&0xF0000000 \| ($addr_{0}$<< 2)	J 3
jr	rs	Jump Register	PC = rs	R 0 / 8
jalr	rs	Jump And Link Register	$ra = PC + 8; PC = rs	R 0 / 9
beq	rt, rs, imm	Branch if Equal	if (rs == rt) PC += 4 + ($imm_{\pm}$<< 2)	I 4
bne	rt, rs, imm	Branch if Not Equal	if (rs != rt) PC += 4 + ($imm_{\pm}$<< 2)	I 5
syscall		System Call	c0_cause = 8 << 2; c0_epc = PC; PC = 0x80000080	R 0 / c
lui	rt, imm	Load Upper Imm.	rt = imm << 16	I f
lb	rt, imm(rs)	Load Byte	rt = SignExt(M_1[rs + $imm_{\pm}$])	I 20
lbu	rt, imm(rs)	Load Byte Unsigned	rt = M_1[rs + $imm_{\pm}$] & 0xFF	I 24
lh	rt, imm(rs)	Load Half	rt = SignExt(M_2[rs + $imm_{\pm}$])	I 21
lhu	rt, imm(rs)	Load Half Unsigned	rt = M_2[rs + $imm_{\pm}$] & 0xFFFF	I 25
lw	rt, imm(rs)	Load Word	rt = M_4[rs + $imm_{\pm}$]	I 23
sb	rt, imm(rs)	Store Byte	M_1[rs + $imm_{\pm}$] = rt	I 28
sh	rt, imm(rs)	Store Half	M_2[rs + $imm_{\pm}$] = rt	I 29
sw	rt, imm(rs)	Store Word	M_4[rs + $imm_{\pm}$] = rt	I 2b
ll	rt, imm(rs)	Load Linked	rt = M_4[rs + $imm_{\pm}$]	I 30
sc	rt, imm(rs)	Store Conditional	M_4[rs + $imm_{\pm}$] = rt; rt = atomic ? 1 : 0	I 38