Design of Digital Circuits

Lab 7 Supplement:
Writing Assembly Code

Prof. Onur Mutlu
ETH Zurich

Spring 2019

16 April 2019

What Will We Learn?

- In Lab 7, you will write MIPS Assembly code
- You will use the MARS simulator to run your code
- References
  - H&H Chapter 6
  - Lectures 9 and 10
    - https://safari.ethz.ch/digitaltechnik/spring2019/doku.php?id=schedule
  - MIPS Cheat Sheet
    - https://safari.ethz.ch/digitaltechnik/spring2019/lib/exe/fetch.php?media=mips_reference_data.pdf

An Example of MIPS Assembly Code

- Add all the even numbers from 0 to 10
  - $0 + 2 + 4 + 6 + 8 + 10 = 30$

High-level code

```
int sum = 0;

for(int i = 0;i <= 10;i += 2)
{

  sum += i;

}
```

MIPS assembly

```
# i=$s0; sum=$s1

        addi $s0, $0, 0
        addi $s1, $0, 0
        addi $t0, $0, 12
loop:   beq  $s0, $t0, done
        add  $s1, $s1, $s0
        addi $s0, $s0, 2
        j    loop
done:
```

Recall: Arrays: Code Example

- We first load the base address of the array into a register (e.g., $s0) using lui and ori

High-level code

```
int array[5];

array[0] = array[0] * 2;

array[1] = array[1] * 2;
```

MIPS assembly

```
# array base address = $s0
# Initialize $s0 to 0x12348000
lui   $s0, 0x1234
ori   $s0, $s0, 0x8000

lw    $t1, 0($s0)
sll   $t1, $t1, 1
sw    $t1, 0($s0)
lw    $t1, 4($s0)
sll   $t1, $t1, 1
sw    $t1, 4($s0)
```

Part 1: Simple Program with Limited Set of Instructions

- Write MIPS assembly code to compute the sum $A + (A + 1) + \cdots (B - 1) + B$, given two inputs A and B.

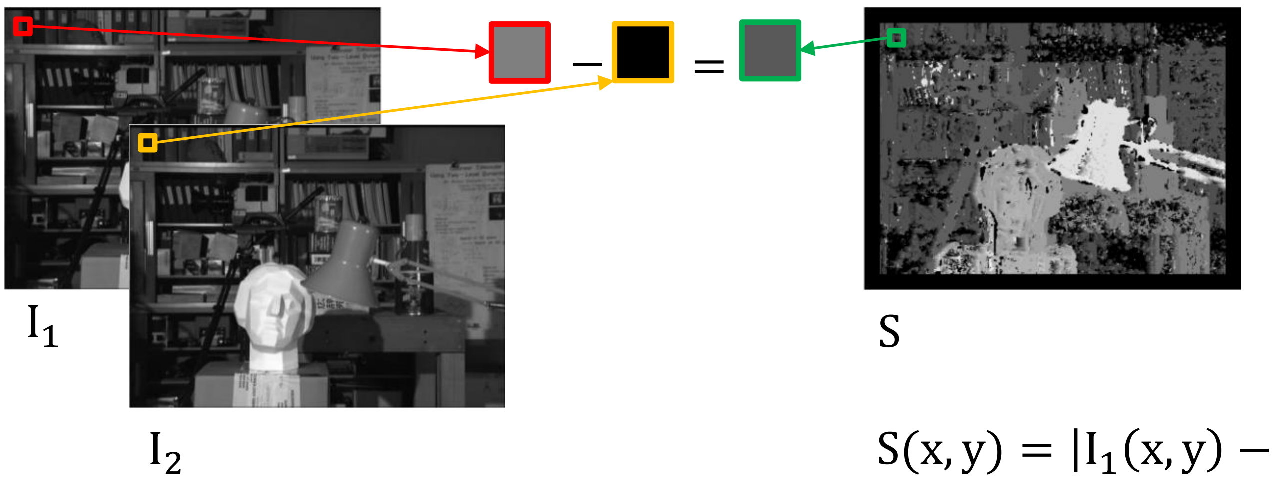
- Example
  - $A = 5, B = 10$ ➔ $S = 5 + 6 + 7 + 8 + 9 + 10 = 45$

- For this exercise, you can use a subset of MIPS instructions: ADD, SUB, SLT, XOR, AND, OR and NOR, which are the instructions supported by the ALU you designed in the previous labs

- Additionally, you are allowed to use J, ADDI and BEQ

Part 2: A More Complex Program (I)

- Write MIPS assembly code to compute the Sum of Absolute Differences (SAD) of two images



$$S(x,y) = |I_1(x,y) - I_2(x,y)|$$

- Hints
  - Recall the function calls and the use of the stack in Lecture 10
  - Read how to implement recursive function calls in H&H 6.4

Part 2: A More Complex Program (II)

- We provide you with a template with **4 TODO parts** that you need to complete.
  - Initializing data in memory.
  - Implement abs_diff() rountine. (from SAD code in manual)
  - Implement the recursive_sum() routine. (from SAD code in manual)
  - Complete the main function to do the corresponding function calls.

- For some sections, you can choose between using our code or writing your own.
  - No extra credit for writing your own code.
  - But it will be a good learning experience.

Last Words

- In this lab, you will do what a compiler does: transforming high level code to MIPS assembly
- Exercise 1: Write simple code and get familiar with the MARS simulator
- Exercise 2: Sum of Absolute Differences of two images
- In the report, you will compute Sum of Absolute Differences of two colored images.

Design of Digital Circuits
Lab 7 Supplement:
Writing Assembly Code

Prof. Onur Mutlu
ETH Zurich
Spring 2019
16 April 2019

Backup Slides

MIPS R-Type Instructions

ADD

Description:	Add two registers and store the result in a register $d.
Operation:	$d = $s + $t; advance_pc (4);
Syntax:	add $d, $s, $t

SUB

Description:	Subtract $t from $s and store the result in $d.
Operation:	$d = $s - $t; advance_pc (4);
Syntax:	sub $d, $s, $t

SLT

Description:	If $s is less than $t, $d is set to one. $d gets zero otherwise.
Operation:	if $s < $t: $d = 1; advance_pc (4); else: $d = 0; advance_pc (4);
Syntax:	slt $d, $s, $t

XOR

Description:	Exclusive or of $s and $t and store the result in $d.
Operation:	$d = $s ^ $t; advance_pc (4);
Syntax:	xor $d, $s, $t

AND

Description:	Bitwise and of $s and $t and store the result in the register $d.
Operation:	$d = $s & $t; advance_pc (4);
Syntax:	and $d, $s, $t

OR

Description:	Bitwise logic or of $s and $t and store the result in $d.
Operation:	$d = $s | $t; advance_pc (4);
Syntax:	or $d, $s, $t

MIPS I-Type Instructions

ADDI

Description:	Add sign-extended immediate to register $s and store the result in $t.
Semantics:	$t = $s + imm; PC=PC+4;
Syntax:	addi $t, $s, imm

BEQ

Description:	Branch if the contents of $s and $t are equal.
Semantics:	if $s == $t: advance_pc (offset << 2)); else: PC=PC+4;
Syntax:	beq $s, $t, offset

MIPS J-Type Instructions

Description:	Jump to the address.	
Semantics:	PC = nPC; nPC = (PC & 0xf0000000) | (target << 2);	
Syntax:	j target	J