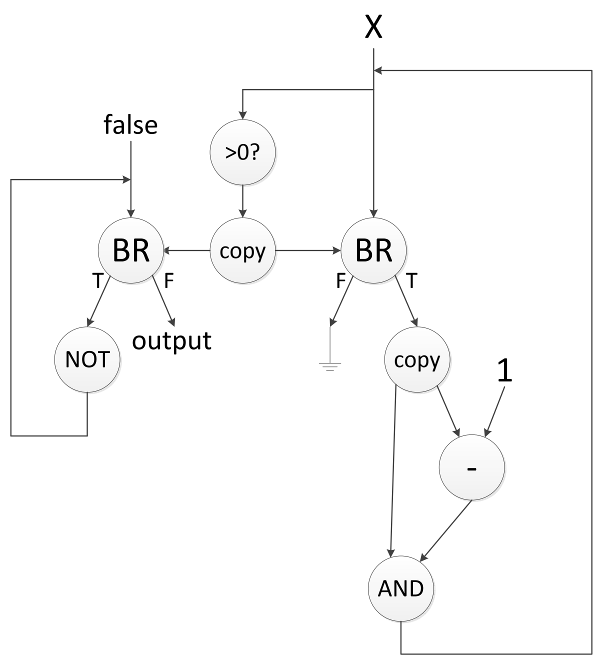
CS 3330 Computer Architecture, Spring 2020
HW 1: Instruction Set Architecture

Instructor: Prof. Samira Khan
TAs: Amel Fatima, Sihang Liu, Korakit Seemakhupt, Yasas Senevirathne, Yizhou Wei

Assigned: Jan 21, 2020
Due: **Jan 30, 2020**

1 Dataflow [40 points]

Here is a dataflow graph representing a dataflow program:



The following is a description of the nodes used in the dataflow graph:

-	subtracts right input from left input
AND	bit-wise AND of two inputs
NOT	the boolean negation of the input (input and output are both boolean)
BR	passes the input to the appropriate output corresponding to the boolean condition
copy	passes the value from the input to the two outputs
>0?	true if input greater than 0
X Y →↓ Z	Initially Z = X then Z = Y

Note that the input X is a non-negative integer.

What does the dataflow program do? Specify clearly in less than 15 words.

2 ISA vs. Microarchitecture [16 points]

Classify the following attributes of a machine as either a property of its microarchitecture or ISA:

1. The machine has 10 ALU unites.
2. There are 32 general purpose registers.
3. The machine dynamically powers down a core when not used.
4. The ADD instruction can only take memory addresses as inputs.
5. There is an instruction called POW which returns the power of 2.
6. The die has heterogeneous core.
7. The last level cache is 4MB.
8. Program counter is always found at register IP.

3 The MIPS ISA [40 points]

3.1 Warmup: Computing a Fibonacci Number [15 points]

The Fibonacci number $F(n)$ is recursively defined as $F(n) = F(n-1) + F(n-2)$, where $F(1) = 1$ and $F(2) = 1$. So, $F(3) = F(2) + F(1) = 1 + 1 = 2$, and so on. Write the MIPS assembly for the fib(n) function, which computes the Fibonacci number $F(n)$:

```
int fib(int n)
{
  int a = 0;
  int b = 1;
  int c = a + b;
  while (n > 1) {
    c = a + b;
    a = b;
    b = c;
    n--;
}
return c; }
```

Remember to follow MIPS calling convention and its register usage (just for your reference, you may not need to use all of these registers):

- The argument n is passed in register $4.
- The result (i.e., c) should be returned in $2.
- $8 to $15 are caller-saved temporary registers.
- $16 to $23 are callee-saved temporary registers.
- $29 is the stack pointer register.
- $31 stores the return address.

A summary of the MIPS ISA is provided at the end of this handout, and a MIPS reference sheet is available at `https://www.cs.virginia.edu/~smk9u/CS3330S20/mips_reference_data.pdf`. The MIPS architecture reference manual is also available at `https://www.cs.virginia.edu/~smk9u/CS3330S20/mips_r4000_users_manual.pdf`.

3.2 MIPS Assembly for REP MOVSB[25 points]

Recall from lecture that MIPS is a Reduced Instruction Set Computing (RISC) ISA. Complex Instruction Set Computing (CISC) ISAs—such as Intel's x86—often use one instruction to perform the function of many instructions in a RISC ISA. Here you will implement the MIPS equivalent for a single Intel x86 instruction, REP MOVSB, which we will specify here[1].

The REP MOVSB instruction uses three fixed x86 registers: ECX (count), ESI (source), and EDI (destination). The "repeat" (REP) prefix on the instruction indicates that it will repeat ECX times. Each iteration, it moves one byte from memory at address ESI to memory at address EDI, and then increments both pointers by one. Thus, the instruction copies ECX bytes from address ESI to address EDI.

1. Write the corresponding assembly code in MIPS ISA that accomplishes the same function as this instruction. You can use any general purpose register. Indicate which MIPS registers you have chosen to correspond to the x86 registers used by REP MOVSB. Try to minimize code size as much as possible.
2. What is the size of the MIPS assembly code you wrote in (1), in bytes? How does it compare to REP MOVSB in x86 (note: REP MOVSB occupies 2 bytes)?
3. Assume the contents of the x86 register file are as follows before the execution of the REP MOVSB:
   `EAX: 0xccccaaaa`
   `EBP: 0x00002222`
   `ECX: 0xFEE1DEAD`
   `EDX: 0xfeed4444`
   `ESI: 0xdecaffff`
   `EDI: 0xdeaddeed`
   `EBP: 0xe0000000`
   `ESP: 0xe0000000`

   Now, consider the MIPS assembly code you wrote in (1). How many total instructions will be executed by your code to accomplish the same function as the single REP MOVSB in x86 accomplishes for the given register state?
4. Assume the contents of the x86 register file are as follows before the execution of the REP MOVSB:
   `EAX: 0xccccaaaa`
   `EBP: 0x00002222`
   `ECX: 0x00000000`
   `EDX: 0xfeed4444`
   `ESI: 0xdecaffff`
   `EDI: 0xdeaddeed`
   `EBP: 0xe0000000`
   `ESP: 0xe0000000`
   Now, answer the same question in (3) for the above register values.

4 Research Paper Summary [20 points]

Please read the following handout on how to write critical reviews. We will give out extra credit that is worth 0.5

Write a half-page summary for the following paper: Onur Mutlu, "Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions", Invited Article, 2018. `https://arxiv.org/pdf/1802.00320.pdf`

[1]The REP MOVSB instruction is actually more complex than what we describe. For those who are interested, please take a look at the Intel architecture manual.

5 Handin

You should electronically hand in your homework (in pdf format) to Collab.

6 MIPS Instruction Summary

Opcode	Example Assembly	Semantic
add	add $1, $2, $3	$1 = $2 + $3
sub	sub $1, $2, $3	$1 = $2 - $3
add immediate	addi $1, $2, 100	$1 = $2 + 100
add unsigned	addu $1, $2, $3	$1 = $2 + $3
subtract unsigned	subu $1, $2, $3	$1 = $2 - $3
add immediate unsigned	addiu $1, $2, 100	$1 = $2 + 100
multiply	mult $2, $3	hi,lo = $2 * $3
multiply unsigned	multu $2, $3	hi,lo = $2 * $3
divide	div $2, $3	lo = $2 / $3, hi = $2 mod $3
divide unsigned	divu $2, $3	lo = $2 / $3, hi = $2 mod $3
move from hi	mfhi $1	$1 = hi
move from low	mflo $1	$1 = lo
and	and $1, $2, $3	$1 = $2 & $3
or	or $1, $2, $3	$1 = $2 | $3
and immediate	andi $1, $2, 100	$1 = $2 & 100
or immediate	ori $1, $2, 100	$1 = $2 | 100
shift left logical	sll $1, $2, 10	$1 = $2 << 10
shift right logical	srl $1, $2, 10	$1 = $2 >> 10
load word	lw $1, 100($2)	$1 = memory[$2 + 100]
store word	sw $1, 100($2)	memory[$2 + 100] = $1
load upper immediate	lui $1, 100	$1 = 100 << 16
branch on equal	beq $1, $2, label	if ($1 == $2) goto label
branch on not equal	bne $1, $2, label	if ($1 != $2) goto label
set on less than	slt $1, $2, $3	if ($2 < $3) $1 = 1 else $1 = 0
set on less than immediate	slti $1, $2, 100	if ($2 < 100) $1 = 1 else $1 = 0
set on less than unsigned	sltu $1, $2, $3	if ($2 < $3) $1 = 1 else $1 = 0
set on less than immediate	sltui $1, $2, 100	if ($2 < 100) $1 = 1 else $1 = 0
jump	j label	goto label
jump register	jr $31	goto $31
jump and link	jal label	$31 = PC + 4; goto label