CS61C Fall 2012 – 3 – The Formation of Integers and MIPS

The Formation of Integers

Idea	Implementation	Pros	Cons
Unsigned	**Start 0 as 0000 0000. Make 1 into 0000 0001. Count upwards.**	**Continuous.**	**No negative numbers.**
Sign/Magnitude	**Sign is first bit (1 = -, 0 = +) Other bits are like unsigned.**	**Has negative numbers.**	**Not continuous.**
One's complement	**If first bit zero, read unsigned. Otherwise, flip all bits and read negative unsigned.**	**Continuous, mostly.**	**2 zeroes.**
Two's complement	**If first bit zero, read unsigned. Otherwise, flip all bits and add 1. Read negative unsigned.**	**Completely continuous. 1 Zero.**	**1 more negative number. Is this really a con / unavoidable?**

Translate the following:		
<u>To Base 8</u> 10_{10} **= 12_8** 77_{10} **= 115_8** 64_7 **= 56_8** <u>To Base 16 (Unsigned)</u> $0000\ 0000_2$ **= 0_{16}** $1011\ 1000_2$ **= $B8_{16}$** $1010\ 1001_2$ **= $A9_{16}$** $1111\ 1110_2$ **= FE_{16}**	<u>To Base 10 (2's C for Binary)</u> 0_{100} **= 0_{10}** 211_3 **= 22_{10}** $0F_{16}$ **= 15_{10}** $0000\ 0000_2$ **= 0_{10}** $0010\ 0000_2$ **= 32_{10}** $1000\ 0000_2$ **= -128_{10}** $1111\ 1100_2$ **= -4_{10}** $1111\ 1111_2$ **= -1_{10}**	<u>To Binary (Use 2's C)</u> 0_{10} **= $0000\ 0000_2$** 15_{10} **= $0000\ 1111_2$** 128_{10} **= Impossible with 8 bits** -18_{10} **= $1110\ 1110_2$** -128_{10} **= $1000\ 0000_2$** FA_{16} **= $1111\ 1010_2$** 364_8 **= $1111\ 0100_2$** 3213_4 **= $1110\ 0111_2$**

MIPS

The Stored Program Concept

- All programs (instructions) are just data represented by combinations of bytes!
- Any block of memory can be code; self-modifying code possible (it's likely system will protect against this)
- The Program Counter (PC) - special register (not directly accessible), holds a pointer to current instruction.

Instruction Formats

R-Instruction format (register-to-register). *Examples: addu, and, sll, jr*

op code	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

See green sheet to see what registers are read from and what is written to

I-Instruction Format (register immediate) Examples: addiu, andi, bne

op code	rs	rt	Immediate
6 bits	5 bits	5 bits	16 bits

Note: Immediate is 0 or sign-extended depending on instruction (see green sheet)

J-Instruction Format (jump format) For j and jal

op code	Address
6 bits	26 bits

MIPS Addressing Modes

- We have several **addressing modes** to access memory (immediate not listed):
  - **Base displacement addressing**: Adds immediate to register value to create memory address (used for lw, lb, sw, sb)
  - **PC-relative addressing**: Uses PC (actually current PC plus four) and adds I-value of instruction (multiplied by 4) to create address (used by I-format branching instructions like beq, bne)
  - **Pseudodirect addressing**: Uses upper four bits of PC + 4 and concatenates a 26-bit value from instruction (with implicit 00 lowest bits) to make 32-bit address (used by J-format instructions)
  - **Register Addressing:** Uses the value in a register as memory (jr)

1) Write MIPS code that will do n mod k where n is the first argument and k is the second.

C Code	MIPS Code

int mod(int n, int k) {
  loop:
  if (n < k){ goto end;}
  n = n – k;
  goto loop;
  end:
  return n;
}
	mod:
	loop: slt $a0 $a1 $t0
	  bne $t0 $0 end
	  sub $a0 $a0 $a1
	  j loop
	end: add $v0 $a0 $0
	  jr $ra

2) Write MIPS code that will do n^k where n is the first argument and k is the second. Do not use MIPS' mult.

C Code	MIPS Code		

int multiply(int n, int k) {
  if (k == 0) {return 0;}
  return n + multiply(n, k-1);
 	
}
 	
int power(int n, int k) {
  if (k == 0) {return 1;}
  return multiply(n, power(n, k-1));
}
	multiply:
	  addi $sp $sp -4
	  sw $ra 0($sp)
	  bne $a1 $0 recurseMult
	  add $v0 $0 $0
	  addi $sp $sp 4
	  jr $ra
	recurseMult:
	  addi $a1 $a1 -1
	  jal multiply
		add $v0 $v0 $a0
		  lw $ra 0($sp)
		  addi $sp $sp 4
		  jr $ra
		power:
		  addi $sp $sp -4
		  sw $ra 0($sp)
		  bne $a1 $0 recursePow
		  addi $v0 $0 1
		  addi $sp $sp 4
			jr $ra
			recursePow:
			  addi $a1 $a1 -1
			  jal power
			  add $a1 $v0 $0
			  jal multiply
			  lw $ra 0($sp)
			  addi $sp $sp 4
			  jr $ra

3) Write MIPS code that will compute the nth Fibonacci number.

C Code	MIPS Code	

int fibonacci(int n) {
  if (n == 0) { return 0;}
  if (n == 1) { return 1;}
  int f1 = fibonacci (n – 1);
  int f2 = fibonacci (n – 2);
  return f1 + f2;
 	
}
	fibonacci:
	  addi $sp $sp -12
	  sw $s1 8($sp)
	  sw $s0 4($sp)
	  sw $ra 0($sp)
	  add $s0 $a0 $0
	  bne $s0 $0 second
	  add $v0 $0 $0
	  j exit
	second:
	  addi $t1 $0 1
	  bne $s0 $t1 fibCase
	  addi $v1 $0 1
	  j exit
		fibCase:
		  addi $a0 $s0 -1
		  jal fibonacci
		  add $s1 $v0 $0
		  addi $a0 $s0 -2
		  jal fibonacci
		  add $v0 $v0 $s1
		exit:
		  lw $s1 8($sp)
		  lw $s0 4($sp)
		  lw $ra 0($sp)
		  addi $sp $sp 12
		  jr $ra