CS61C Fall 2013 – 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.	Fixed positive slope.	2 zeroes. Not continuous.
Two's complement	If first bit zero, read unsigned. Otherwise, flip all bits and add 1. Read negative unsigned.	Fixed positive slope. 1 Zero.	1 more negative number. Not continuous.

Differences in Representation

Decimal	0	-1	15	-16	MAX	MIN
Unsigned	0b00000	N/A	0b01111	N/A	0b11111 (31)	0b00000 (0)
Sign / Mag.	0b00000 0b10000	0b10001	0b01111	N/A	0b01111 (15)	0b11111 (-15)
One's Comp.	0b00000 0b11111	0b11110	0b01111	N/A	0b01111 (15)	0b10000 (-15)
Two's Comp.	0b00000	0b11111	0b01111	0b10000	0b01111 (15)	0b10000 (-16)

Changing Bases

Translate the following:		
To Base 8 10_{10} = 12_8 77_{10} = 115_8 64_7 = 56_8 To Base 16 (Unsigned) $0000\ 0000_2$ = 0_{16} $1011\ 1000_2$ = $B8_{16}$ $1010\ 1001_2$ = $A9_{16}$ $1111\ 1110_2$ = FE_{16}	To Base 10 (2's C for Binary) 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}	To Binary (Use 2's C) 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 = $1110\ 0111_2$

Intro to 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.
- For recursion: adjust the stack pointer ($sp) to save return address ($ra) and other registers (ex: $s0)

C To MIPS

C Code	MIPS Code

int a = 5, b = 10;
if (a + a == b) {
   a = 0;
} else {
   b = a - 1;
}
	addiu $s0, $0, 5
	addiu $s1, 0, 10
	add $t0, $s0, $s0
	bne $t0, $s1, else
	add $s0, $0, $0
	j exit
	
	else:
	   addiu $s1, $s0, -1
	exit:
	   #done

C To MIPS: Recursion

C Code	MIPS Code

int sum(int n) {
  return n ? n + sum(n – 1) : 0;
}
// Use recursion in your MIPS!
	sum:
	addi $sp, $sp, -8 # allocate space on stack
	sw $ra, 0($sp) # store the return address
	sw $a0, 4($sp) # store the argument
	slti $t0, $a0, 1 # check if n > 0
	beq $t0, $0, recurse # n > 0 case
	add $v0, $0, $0 # start return value to 0
	addi $sp, $sp, 8 # pop 2 items off stack
	jr $ra # return to caller
	
	recurse:
	addi $a0, $a0, -1 # calculate n-1
	jal sum # recursively call sum(n-1)
	lw $ra, 0($sp) # restore saved return address
	
	lw $a0, 4($sp) #restore saved argument
	addi $sp, $sp, 8 #pop 2 items off stack
	add $v0, $a0, $v0 #calculate n + sum(n-1)
	jr $ra #return to caller

MIPS To C

C Code	MIPS Code

int a = 0;
int b = 1;
int c = 30;
while (a != c) {
  b = b << 1;
  a++;
}
 	
What does this code do?
> Calculates 2^30 in variable b.
	addi $s0, $0, 0
	addi $s1, $0, 1
	addi $t0, $0, 30
	loop: beq $s0, $t0, done
	    sll $s1, $s1, 1
	    addi $s0, $s0, 1
	    j loop
	done: # done