Lecture # 40

- Transition to RISC Assembly

MIPS registers

register	assembly name	Comment
r0	$zero	Always 0
r1	$at	Reserved for assembler
r2-r3	$v0-$v1	Stores results
r4-r7	$a0-$a3	Stores arguments
r8-r15	$t0-$t7	Temporaries, not saved
r16-r23	$s0-$s7	Contents saved for use later
r24-r25	$t8-$t9	More temporaries, not saved
r26-r27	$k0-$k1	Reserved by operating system
r28	$gp	Global pointer
r29	$sp	Stack pointer
r30	$fp	Frame pointer
r31	$ra	Return address

	Memory address label	Operation	Addressing or data information	
Assembler directive		ORIGIN	100	
Statements that generate machine instructions		LD	R2, N	
		CLR	R3	
		MOV	R4, #NUM1	; Load starting address
	LOOP:	LD	R5, (R4)	; Read first element
		ADD	R3, R3, R5	
		ADD	R4, R4, #4	
		SUB	R2, R2, #1	
		BGT	R2, R0, LOOP	
		ST	R3, SUM	
		next instruction		
Assembler directives		ORIGIN	200	
	SUM:	RESERVE	4	
	N:	DATAWORD	150	
	NUM1:	RESERVE	600	
		END		

Move R4, #NUM1

In many RISC-type processors, one general-purpose register is dedicated to holding a constant value zero. Usually, this is register R0. Its contents cannot be changed by a program instruction. We will assume that R0 is used in this manner in our discussion of RISC-style processors. Then, the above Move instruction can be implemented as

```
Add     R4, R0, #NUM1
```

It is often the case that Move is provided as a *pseudoinstruction* for the convenience of programmers, but it is actually implemented using the Add instruction.

BGT R4, R5, LOOP

It compares the contents of registers R4 and R5, without changing the contents of either register. Then, it causes a branch to LOOP if the contents of R4 are greater than the contents of R5.

Question # 1

A.7 [20/20] <A.2, A.9> Consider the following fragment of C code:

```
for (i = 0; i <= 100; i++)
{ A[i] = B[i] + C; }
```

Assume that A and B are arrays of 64-bit integers, and C and i are 64-bit integers. Assume that all data values and their addresses are kept in memory (at addresses 1000, 3000, 5000, and 7000 for A, B, C, and i, respectively) except when they are operated on. Assume that values in registers are lost between iterations of the loop.

a. [20] <A.2, A.9> Write the code for MIPS.

b. [20] <A.2> Write the code for x86.

c. Suppose ALU operations take 1 cycle and memory operations take 3 cycles. Calculate total cycles this program takes from start to finish. Consider all repeated instructions.