1 (12%) Fig.1is the flow chart and Fig.2 is the architectures of binary multiplier.

1.1 (4%) Give the values of Q. (binary code)

1.2 (8 %) Give the second and fifth partial products of CAQ after shift right in Table 1. (binary code)

Table 1

Multiplicand B = 10110

	(Carry) C	(5 bits) A	(5 bits) Q	P
Multiplier in Q	0	00000		
Qo = 1; add B				
First partial product				
Shift right CAQ				
Qo = 1 ; add B				
Second partial product				
Shift right C AQ				
Qo = 0 ; shift right CAQ				
Qo = 0 ; shift right CAQ				
Qo = 1 ; add B				
Fifth partial product				
Shift right CAQ				

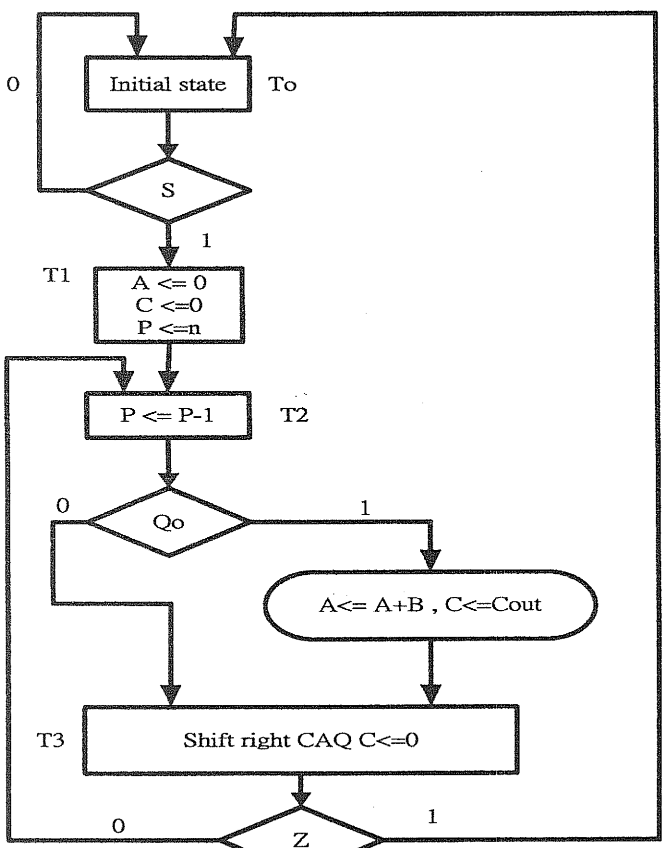

Fig.1

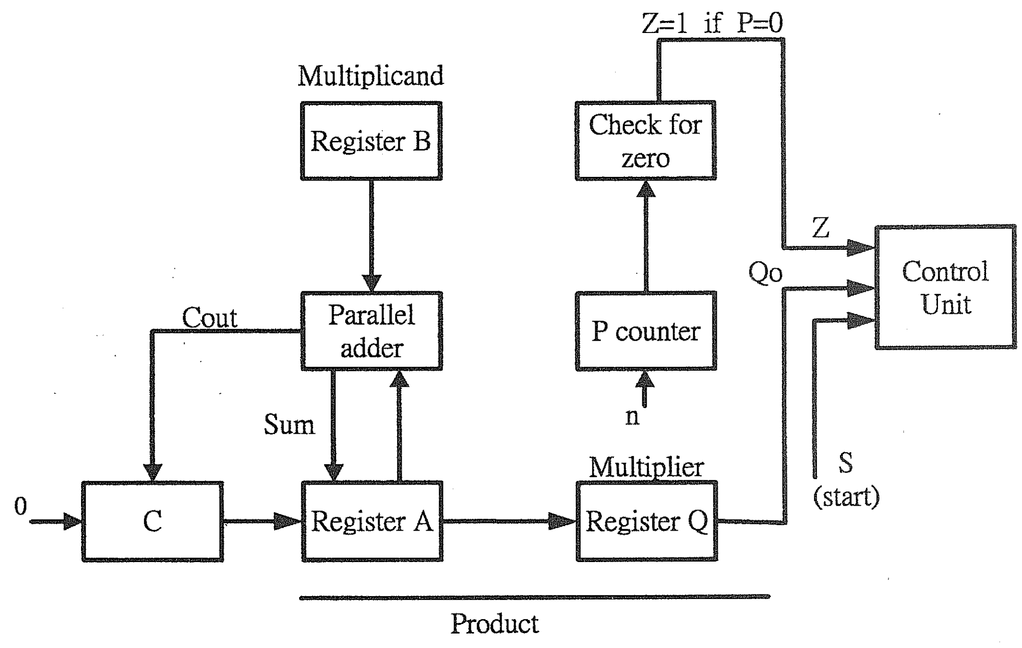

Fig 2

2. (10%) Derive the correct floating-point representation for the decimal numbers -13.25 using the 32-bit IEEE 754 floating-point standard and give the largest positive number.

3. (9%)Explain the following terms:
   3.1 Addressing modes
   3.2 DMA (Direct Memory Access)
   3.3 Write back vs. Write through

4. (9%) A memory data register DR can transfer 32-bit words to M in a single clock cycle. The data items to be stored can be 4, 8, 16, or 32 bits long, and short items are always sign-extended to 32 bits for transmission to M. A 2-bit flag S in the CPU is set to 00, 01, 10, or 11 to indicate a data size of 4, 8, 16, or 32 bits, respectively. Design an efficient logic circuit at the register level to implement the sign extension.

5. (10%) Calculate X x Y (X=101011, Y=100011)
   5.1 By Robertson multiplication algorithm :

$$x = -2^{n-1} x_{n-1} + \sum_{i=0}^{n-2} 2^i x_i$$

   5.2 By Booth's multiplication algorithm.

6. (15%) Compare CISC and RISC processors in terms of instruction formats, clock cycle time, clock cycles per instruction, performance, and power consumption. Please explicitly state the reasons of each comparison.

7. (25%) Assume that there is no multiplication in the MIPS instruction set.
   Please implement the function "unsigned int sum(unsigned int n)" which returns
   the value of "1 + 2 + . . . + n".
   7.1 Write the C code with a while loop. (5%)
   7.2 Write the corresponding MIPS code in 7.1. (7%)
   7.3 Write the C code with recursive procedural calls. (5%)
   7.4 Write the corresponding MIPS code in 7.3. (8%)

8. (10%) Assume an instruction cache miss rate of 2% and a data cache miss rate of 4%. If a machine has a CPI of 2 without any memory stalls and the miss penalty is 40 cycles for all misses, determine how much faster a machine would run with a perfect cache that never missed. Assume 36% of instructions are loads/stores. (Assume that n is mapped to the argument register \$a0.)

MIPS Instruction Set Quick Reference

Jumps And Branches (Note: One Delay Slot)		
B	off18	PC += off18$^{\pm}$
BAL	off18	RA = PC + 8, PC += off18$^{\pm}$
BEQ	Rs, Rt, off18	if Rs = Rt, PC += off18$^{\pm}$
BEQZ	Rs, off18	if Rs = 0, PC += off18$^{\pm}$
BGEZ	Rs, off18	if Rs ≥ 0, PC += off18$^{\pm}$
BGEZAL	Rs, off18	RA = PC + 8; if Rs ≥ 0, PC += off18$^{\pm}$
BGTZ	Rs, off18	if Rs > 0, PC += off18$^{\pm}$
BLEZ	Rs, off18	if Rs ≤ 0, PC += off18$^{\pm}$
BLTZ	Rs, off18	if Rs < 0, PC += off18$^{\pm}$
BLTZAL	Rs, off18	RA = PC + 8; if Rs < 0, PC += off18$^{\pm}$
BNE	Rs, Rt, off18	if Rs ≠ Rt, PC += off18$^{\pm}$
BNEZ	Rs, off18	if Rs ≠ 0, PC += off18$^{\pm}$
J	addr28	PC = $PC_{31:28}$ :: addr28$^{\varnothing}$
JAL	addr28	RA = PC + 8; PC = $PC_{31:28}$ :: addr28$^{\varnothing}$
JALR	Rd, Rs	Rd = PC + 8; PC = Rs
JR	Rs	PC = Rs

Condition Testing and Conditional Move Operations		
MOVN	Rd, Rs, Rt	if Rt ≠ 0, Rd = Rs
MOVZ	Rd, Rs, Rt	if Rt = 0, Rd = Rs
SLT	Rd, Rs, Rt	Rd = (Rs$^{\pm}$ < Rt$^{\pm}$) ? 1 : 0
SLTI	Rd, Rs, const16	Rd = (Rs$^{\pm}$ < const16$^{\pm}$) ? 1 : 0
SLTIU	Rd, Rs, const16	Rd = (Rs$^{\varnothing}$ < const16$^{\varnothing}$) ? 1 : 0
SLTU	Rd, Rs, Rt	Rd = (Rs$^{\varnothing}$ < Rt$^{\varnothing}$) ? 1 : 0

Default C Calling Convention (O32)

Stack Management
- The stack grows down.
  - Subtract from $sp to allocate local storage space.
  - Restore $sp by adding the same amount at function exit.
- The stack must be 8-byte aligned.
  - Modify $sp only in multiples of eight.

Function Parameters
- Every parameter smaller than 32 bits is promoted to 32 bits.
- First four parameters are passed in registers $a0–$a3.
  - 64-bit parameters are passed in register pairs:
    - Little-endian mode: $a1:$a0 or $a3:$a2.
    - Big-endian mode: $a0:$a1 or $a2:$a3.
- Every subsequent parameter is passed through the stack.
  - First 16 bytes on the stack are not used.
  - Assuming $sp was not modified at function entry:
    - The 1st stack parameter is located at 16($sp).
    - The 2nd stack parameter is located at 20($sp), etc.
  - 64-bit parameters are 8-byte aligned.

Return Values
- 32-bit and smaller values are returned in register $v0.
- 64-bit values are returned in registers $v0 and $v1:
  - Little-endian mode: $v1:$v0.
  - Big-endian mode: $v0:$v1.

Registers		
0	zero	Always equal to zero
1	at	Assembler temporary; used by the assembler
2-3	v0-v1	Return value from a function call
4-7	a0-a3	First four parameters for a function call
8-15	t0-t7	Temporary variables; need not be preserved
16-23	s0-s7	Function variables; must be preserved
24-25	t8-t9	Two more temporary variables
26-27	k0-k1	Kernel use registers; may change unexpectedly
28	gp	Global pointer
29	sp	Stack pointer
30	fp/s8	Stack frame pointer or subroutine variable
31	ra	Return address of the last subroutine call

Arithmetic Operations		
ADD	Rd, Rs, Rt	Rd = Rs + Rt (overflow trap)
ADDI	Rd, Rs, const16	Rd = Rs + const16$^{\pm}$ (overflow trap)
ADDIU	Rd, Rs, const16	Rd = Rs + const16$^{\pm}$
ADDU	Rd, Rs, Rt	Rd = Rs + Rt
CLO	Rd, Rs	Rd = CountLeadingOnes(Rs)
CLZ	Rd, Rs	Rd = CountLeadingZeros(Rs)
LA	Rd, label	Rd = Address(label)
LI	Rd, imm32	Rd = imm32
LUI	Rd, const16	Rd = const16 << 16
MOVE	Rd, Rs	Rd = Rs
NEGU	Rd, Rs	Rd = –Rs
SEB[R2]	Rd, Rs	Rd = $Rs_{7:0}^{\pm}$
SEH[R2]	Rd, Rs	Rd = $Rs_{15:0}^{\pm}$
SUB	Rd, Rs, Rt	Rd = Rs – Rt (overflow trap)
SUBU	Rd, Rs, Rt	Rd = Rs – Rt

Load and Store Operations		
LB	Rd, off16(Rs)	Rd = mem8(Rs + off16$^{\pm}$)$^{\pm}$
LBU	Rd, off16(Rs)	Rd = mem8(Rs + off16$^{\pm}$)$^{\varnothing}$
LH	Rd, off16(Rs)	Rd = mem16(Rs + off16$^{\pm}$)$^{\pm}$
LHU	Rd, off16(Rs)	Rd = mem16(Rs + off16$^{\pm}$)$^{\varnothing}$
LW	Rd, off16(Rs)	Rd = mem32(Rs + off16$^{\pm}$)
LWL	Rd, off16(Rs)	Rd = LoadWordLeft(Rs + off16$^{\pm}$)
LWR	Rd, off16(Rs)	Rd = LoadWordRight(Rs + off16$^{\pm}$)
SB	Rs, off16(Rt)	mem8(Rt + off16$^{\pm}$) = $Rs_{7:0}$
SH	Rs, off16(Rt)	mem16(Rt + off16$^{\pm}$) = $Rs_{15:0}$
SW	Rs, off16(Rt)	mem32(Rt + off16$^{\pm}$) = Rs
SWL	Rs, off16(Rt)	StoreWordLeft(Rt + off16$^{\pm}$, Rs)
SWR	Rs, off16(Rt)	StoreWordRight(Rt + off16$^{\pm}$, Rs)
ULW	Rd, off16(Rs)	Rd = unaligned_mem32(Rs + off16$^{\pm}$)
USW	Rs, off16(Rt)	unaligned_mem32(Rt + off16$^{\pm}$) = Rs