Q1 a) sw $t1, 20($t0)

- RTL for sw MEM[R[rs] + sign_ext(Im16)] ← R[rt]
  - MEM[R[$t0] + sign_ext(20)] ← R[$t1]
- IF: has PC inc and fetch from instr. memory
  - PC inc has: Add (50ps) (No MUX for ROW - comes in MEM)
  - Inst. mem is 300ps
  - IF takes 300ps for sw.
- ID: has Reg file and sign_ext.
  - Reg file takes 85ps (assuming all regs can be accessed at same time.
  - Sign_ext is 10ps
  - ID is 85 ps for sw.
- Ex: ALU comp, Branch result
  - ALU comp: MUX (15ps) + ALU comp (100 ps) = 115ps
  - Branch comp: shift (5ps) + Add (50ps) = 55ps
  - Ex is 115ps for sw
- MEM: mem access, wb to PC.
  - Mem acc (400ps) ← Mem for sw
  - wb to PC (15ps), PC reg (85 ps),
- wb: wb MUX (15ps) + Reg file (85ps) = 100ps
  - Doesn't matter for sw.
- latency = critical path [IF, ID, Ex, Mem] = 300 + 85 + 115 + 100 = **900ps**

b) Beq $t1, $t0, Label

- RTL: if (R[$t1] = R[$t0]) then PC ← PC + in (sign_ext (label) << 2)
  else PC ← PC + 4.
- IF: 300ps, ID: 85ps, Ex: 15ops, Mem: 15ps + 0ps. = **530ps**

c) R-type Paths:

- for in PC: PC → Add → MUX → PC
- for operation: IM → Reg file → MUX → ALU → MUX → reg ← critical path (600ps)

I-type: longest would be load word due to reg file

- IM → Reg → MUX → ALU → Data memory → MUX → reg ← critical path (1000ps)

J-type: The figure is not able to do jump instructions.
- If it could based on lecture notes.
  CP: PC → Add → MUX → PC.

d) The clock rate should correspond with the stage that takes the longest for all instructions. At a glance, the ones that might take the longest are the If, Ex, Mem stages.
- If reads from the PC val from IM and inc the PC
  - INC: PC → add (could have be PC src in mem stage)
  - Read In is 300ps
- Ex handles calculations and branches
Cal: MUX → ALU 115ps, branches: MUX → ALU → MUX = 130ps.
- Mem: Writes back to PC and W/R from mem.
  - W/R PC: MUX → PC, mem: 400ps.

Thus the longest stage is 400ps and that is clock rate is 400ps.

Q2 a) sw rt Imm(rs)

i) Hardware modifications: none.

ii) Control signals:

RW	AS	AO	MW	PS	MR	MTR
0	1	00	1	0	0	X

iii) Data Path:
- Storing rt: IM → Regs → Cal R[rs] + Sign_ext[Imm] → Write to DM.
- Inc. PC: PC → Add 4 → PC

b) Control signals: RW, AS, AO, MW, PS, MR, MTR ← Control unit [IR [31:26, 5:0]]
Instruction Memory: IR ← IM [PC]
Registers: IP RW==0: RR1 ← IR [25:21], RR2 ← IR[20:16]
ALU: if AS==1 & ALUOP = 00: ALUres ← RD1 + sign_ext [Imm]
DM: if MW==1 & MR==0: M = M[ALUres] ← RD2.
Adder1: Adder1_out ← PC+4
Adder2: Adder2_out ← Adder1_out + SL [sign_ext (Imm)]
PC Adder: If PS==0: PC ← Adder1_out.

Q.3 a) Given data memory used for lw & sw instructions.

So, the fraction is: $\frac{15+15}{100} = \frac{3}{10}$ //

b) The sign-extend circuit needs cycle fraction is:

addi + not + beq + lw + sw

So, the fraction is: $\frac{15+15+35+15+15}{100} = \frac{95}{100}$

$= \frac{19}{20}$ //

Q.4

Instruction class	Instruction Decode	Instruction Fetch	Execution	Register write	Memory	Total
Store word	350	250	150		300	1050 ps
R-type	350	250	150	200		950 ps
Branch	350	250	150			750 ps
Load word	350	250	150	200	300	1250 ps

Ex 4.8.1:

∴ The slowest processor stage is 350 ps. Clock cycle time for pipelined processor is: 350 ps //

∴ The non-pipelined processor cycle time for all stages is:

350 + 250 + 150 + 200 + 300 = 1250 ps //

Ex 4.8.2:

The total latency for the lw instruction is 1250 ps for no-pipelining.

∴ The latency for pipelined processor of an lw instruction:

5 × 350 = 1750 ps //.

Ex 4.8.3:

Pipelining to the five stages is reduced the cycle time to the length of the longest stage. Thus, splitting of the longest stage is most usual way to decrease the cycle time. After splitting, the new cycle is based upon the longest stage. Thus, the longest stage ID would be split into two and the clock cycle is 300 ps.

Ex 4.8.4:
Among the above instructions in the table, only lw and sw instructions utilize the data memory.
∴ Utilization of data memory = lw + sw
= 20 + 15 = 35% //

Ex 4.8.5:
To write the port for register block by processor only utilized two instructions lw and ALU.
∴ Utilization of register block = ALU + lw
= 45 + 20 = 65% //

Ex 4.8.6:
The multi-cycle organization has the same clock cycle as the pipelined organization. From question.

$$X = (5 \times lw) + (4 \times sw) + (4 \times ALU) + (4 \times beq)$$
$$= 5 \times 0.2 + (0.15 + 0.45 + 0.2) \times 4$$
$$= 1 + 3.2$$
$$= 4.2 //$$

∴ The multicycle execution time is 4.2 times pipelined execution time.
The single-cycle execution time is X times pipelined execution time:

$$X = \frac{\text{Total cycle time}}{\text{Cycle time of pipeline}}$$
$$= \frac{1250 \text{ ps}}{350 \text{ ps}} = 3.57 //$$

∴ The single-cycle execution is 3.57 times pipelined execution time.

Q.5 Ex 4.7.1
A unit is needed to sign-extend the 16-bit offset field in the instruction to a 32-bit signed value. The 0-15 bits contain most significant bit 0. Thus, the output of the sign-extend is: 0000000000000000 0000000000010100.

In a jump, the lower 26-bits are taken and it is expanded to 28-bits with a left shift. The control signal jump is the sequence left by 2 bits. Thus, the output of the shift left 2 unit is: 00011000100000000001010000.

Ex 4.7.2:

Value of ALU Op: 10101100011000100000000000010100

Value of instruction: 10101100011000100000000000010100

∴

ALU op [1:0]	Instruction [5:0]
00	010100

Ex 4.7.3:

After execution: PC → PC+4

Path: PC → PC+4 → Branch Mux → Jump Mux → PC.

Ex 4.7.4:

- From the given word, the input values [20-16] are 0010 & [15-11] are 0000. The MUX selects one of the values from 2 or 0. The data output value of Write register MUX is 2 or 0.
- For the ALU MUX, the input sign-extend unit is [15-0]. Append zeros at the MSB side to extend it to 32-bit value and the decimal value is equivalent to 20. The data output value of ALU MUX is 20.
- The data memory is the read data and ALU result. The given instruction is store & it doesn't read any data from memory. The data value of data-memory is [X] for don't care.
- Branch and jump MUXs are used to increment the PC value to fetch the next instruction which is to be executed. ∴ The output value is PC+4.

Ex 4.7.5

From the given data, the syntax is sw $t, offset ($s). Which is to store the value in the register 3 into memory [offset + $s]. The value in register 3 is -3. Store the value 3 into the memory [10100 + 00010] which is memory [20+2].

For the ALU, one of the inputs is to read the register value ie -3.

The other input is sign-extend 16 bit ie 20.
∴ The ALU values **-3 and 20**
For the Add unit, one of the inputs is PC+4 [31-28] & other input 16-bit sign-extend shift left 2 ie 20 [00010100] becomes 80 [01010000]
∴ The values in Add unit are **PC+4 and 80.** (Branch unit)
For the Add unit (PC), one of the inputs to the add unit are PC & 4 to fetch the next instruction.
∴ The add unit value are **PC and 4.**
Ex 4.7.6:
The value for the Read register 1 input is Instruction [25-21]: 00011
The value for the Read register 2 input is Instruction [20-16]: 00010.
For sw instruction, the value of RegDst could be 0 or 1. So, the output of the MUX for the Write register input can be instruction [20-16] : 00010 or instruction [15-11]: 00000 //.
And sw doesn't write on any registers. So, the value for Register Write input is **0** //