NYU Tandon School of Engineering

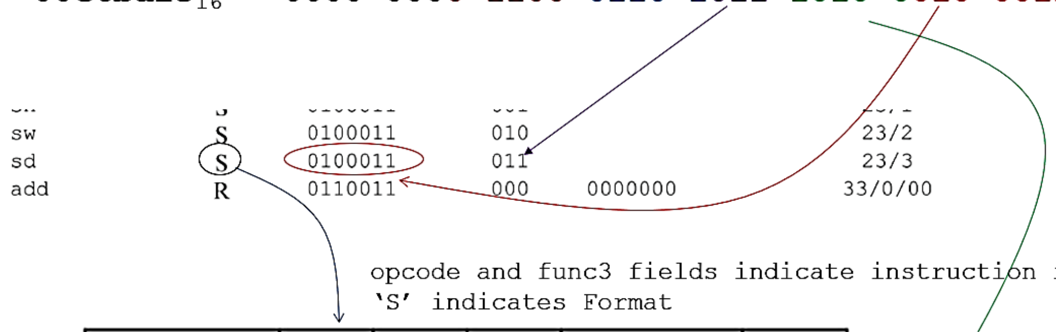
Fall 2022, ECE 6913

Homework Assignment 5 Solutions

---

1. In this exercise, we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word: **`0x00c6ba23`**.

$$00c6ba23_{16} = 0000\ 0000\ 1100\ 0110\ 1011\ 1010\ 0010\ 0011$$



sw	S	0100011	010		23/2
sd	S	0100011	011		23/3
add	R	0110011	000	0000000	33/0/00

opcode and func3 fields indicate instruction is 'sd'
'S' indicates Format

immediate[11:5]	rs2	rs1	funct3	immediate[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

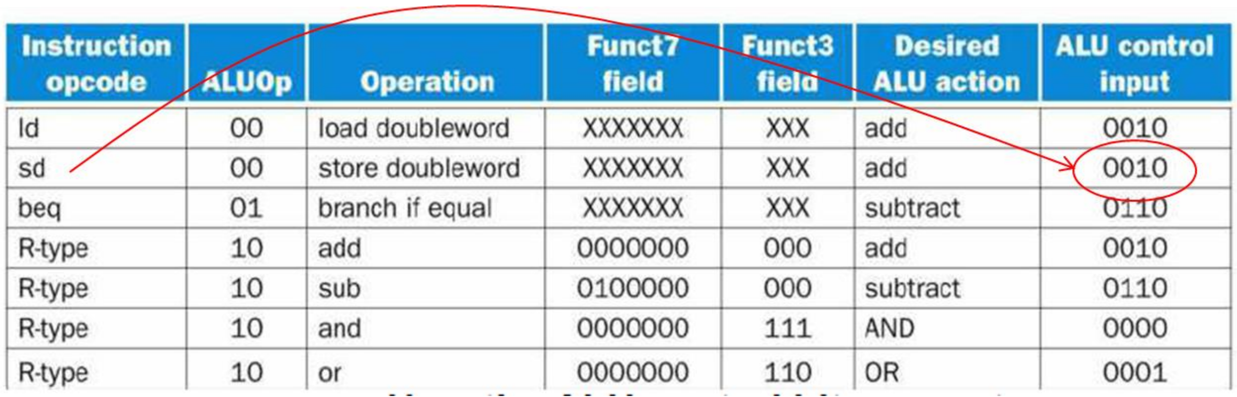
- **The source register fields rs1 & rs2 identify register #s x13 and x12**
- **S format splits the 12 bit immediate field in [31:25] & [11:7] of instruction**
- **So, instruction is `sd x12, 20(x13)`**

1.1 What are the values of the ALU control unit's inputs for this instruction?

ALUop bits for **ld, sd** instructions are **00**

ALU Control line bits are 0010 *[from table in fig 4.12 of text]*

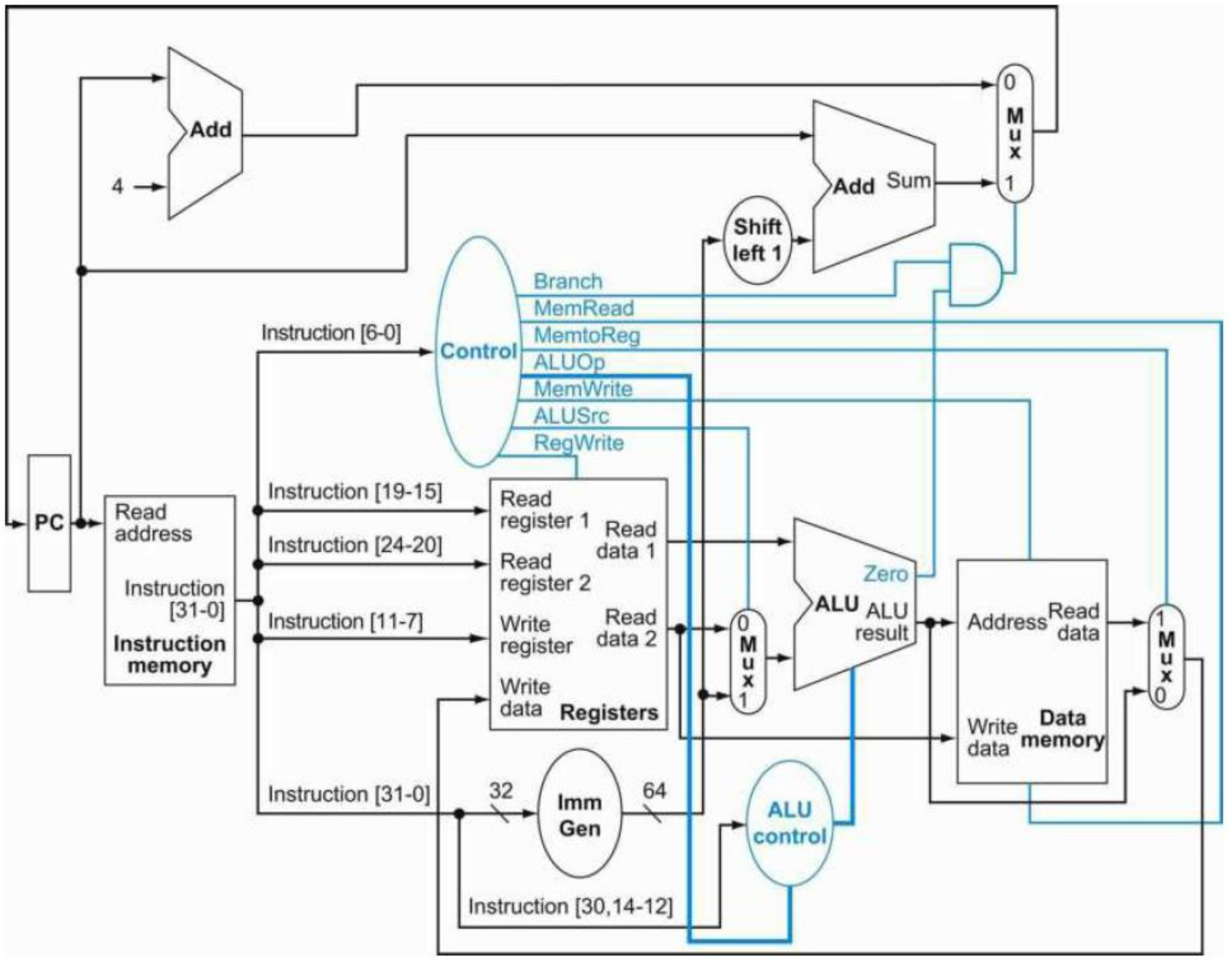
Since the **'add'** operation is executed in the ALU for a sd instruction (add offset in immediate 5 bit field (20) to base address (in rs1), the ALU Control lines take the value **0010**



Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
ld	00	load doubleword	XXXXXXX	XXX	add	0010
sd	00	store doubleword	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

1.2 What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

Since the **sd** instruction does not take any branch, the new PC address after this instruction is executed is **PC+4**



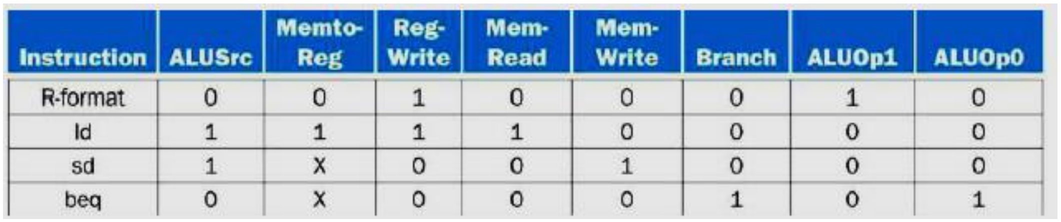
Datapath above from Fig 4.17 in text

1.3 For each mux, show the values of its inputs and outputs during the execution of this instruction. List values that are register outputs at `Reg [xn]`.

The 3 mux units shown in Fig 4.17 are listed in the table below: ALUsrc, PCsrc and MemtoReg

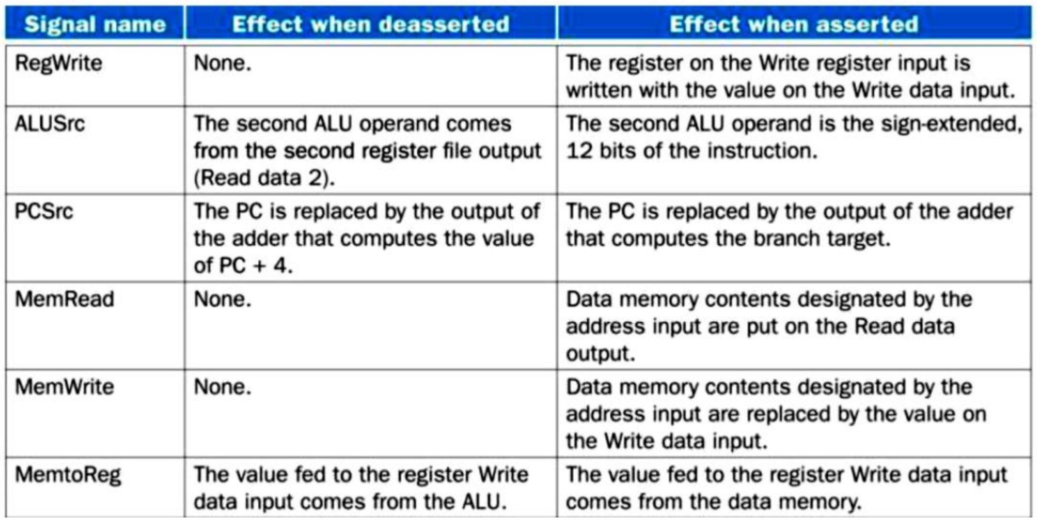
For the **sd** instruction,

ALUsrc is '1' (in Table 4.18 reproduced below) and takes the input to this mux from the sign extend unit (**0x0..014**) - instead of register **rs2** (x12 in this problem), to add to the contents of the base register **rs1** in the ALU to determine the address in memory to store contents of register **rs2**. The decimal number '20' corresponding to the sign extended offset (in hex) = 0x00..014.



Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
ld	1	1	1	1	0	0	0	0
sd	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

Table 4.18 from text *– providing 8 control signals from the Control unit based on the 7-bit opcode of the instruction*



Signal name	Effect when deasserted	Effect when asserted
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, 12 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

Table 4.16 from text *– defining 6 of 8 control signals from the Control unit (not including the ALUop bits).*

PCsrc is '0' since this mux is asserted only for branch instructions

MemtoReg is value is irrelevant (don't care). In table below it is asserted '1' this choice picks data from Data Memory to send to the register file for Write back. Data Memory is in Write mode

(Mem Write =1 in table 4.18 for a sd instruction) so the data at output buffer of Data memory is undefined. This data is routed to write port of Register file array. But Reg Write control bit in Table 4.18 is ‘0’ so undefined data not used in Register File.

Mux	Control input	input 1	input 2	output
ALUsrc	1	**rs2**	0x00..014	0x0..014
PCsrc	0	PC+4	offs sll2	PC + 4
MemtoReg	δ	**rs1**+0x..14	unknown	unknown

1.4 What are the input values for the ALU and the two add units?

From Fig 4.17 below, ALU inputs are **rs1** (Reg[x13]) and **0x00..014** from the mux controlled by **ALUsrc**

Branch adder inputs: **PC** and 0x0..014 shifted by 1 (**0x0..28**)

PC adder inputs: PC and 4

1.5 What are the values of all inputs for the register’s unit?

Read register 1 (**rs1**) input port has the 5-bit instruction field [19-15] identifying the register **x13**, Read register 2 (**rs2**) input port has the 5-bit instruction field [24 – 20] identifying the register **x11.** Write register input port has undefined/irrelevant bits since Reg Write control signal is disabled for the **sd** instruction.

2. Problems in this exercise assume that the logic blocks used to implement a processor's datapath have the following latencies:

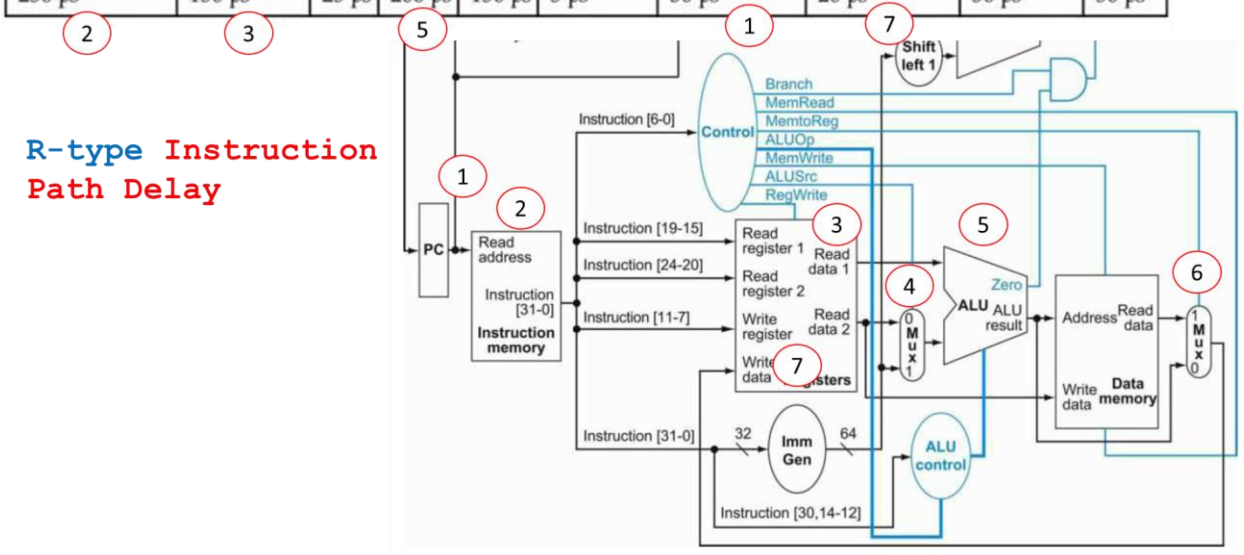
I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps

"Register read" is the time needed after the rising clock edge for the new register value to appear on the output. This value applies to the PC only. "Register setup" is the amount of time a register's data input must be stable before the rising edge of the clock. This value applies to both the PC and Register File.

2.1 What is the latency of an `R-type` instruction (i.e., how long must the clock period be to ensure that this instruction works correctly)?

```
the R-type and I-type instructions both have identical delays (even
though the paths are different):
1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
4. Mux to pick data from either R2 or sign extended bit data from
immediate field of instruction = 25 ps
[note - there is no number given for sign extension delay, so we can
assume it is less than the register file delay of 150 ps]
5. ALU delay: 200 ps
6. Mux for WB to Register File of result from ALU: 25 ps
7. Write back setup time for Register File registers: 20 ps
---------------------------------------------------------------
```
total = 700ps

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps

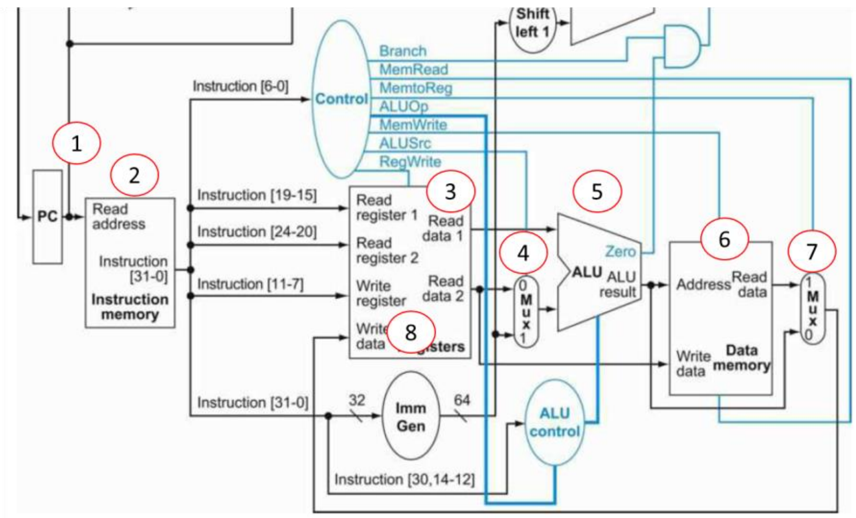


2.2 What is the latency of `ld`? (Check your answer carefully. Many students place extra muxes on the critical path.)

For ld , *item 6 below is new*:
1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
4. Mux to pick data from either R2 or sign extended bit data from immediate field of instruction = 25 ps
[note - number given for sign extension delay of 50 ps is less than the 150 ps for register file read access, so we can assume that mux control signal is designed to fire only when both inputs are ready]
5. ALU delay: 200 ps
6. Read data from Data Mem, whose address provided by ALU: 250 ps
7. Mux for WB *to Register File* of result from Data Mem: 25 ps
8. Write back setup time *for Register File* registers: 20 ps
----
total ld = 950ps

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps

ld Instruction Path Delay

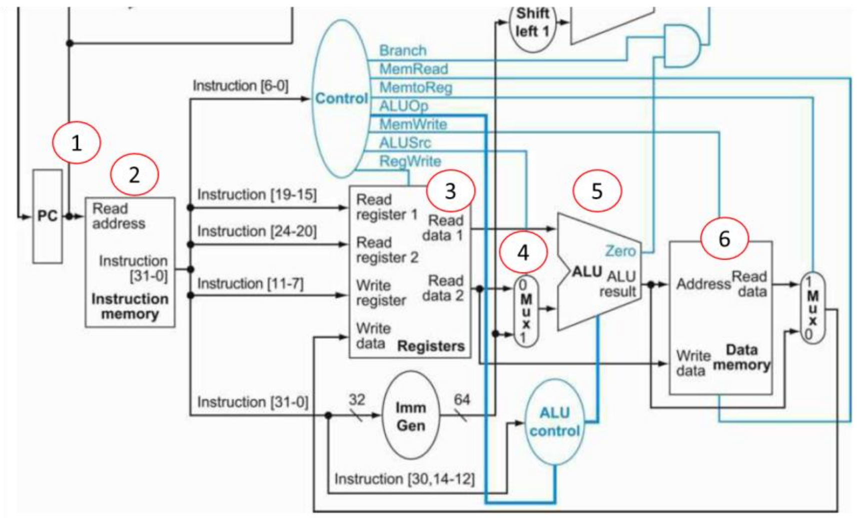


2.3 What is the latency of `sd`? (Check your answer carefully. Many students place extra muxes on the critical path.)
for sd
1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
4. Mux to pick data from either R2 or sign extended bit data from immediate field of instruction = 25 ps
[note - there is no number given for sign extension delay, so we can assume it is less than the register file delay of 150 ps]
5. ALU delay: 200 ps
6. Write Data from R2 to Data Mem: 250 ps

Total for sd: 905 ps

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps

sd Instruction Path Delay



2.4 What is the latency of `beq`?

1. **PC Register Read** delay following rising edge of clock: 30 ps
2. **Instruction Memory**: 250ps
3. Read **Register File**: 150 ps
4. **Mux** to pick data from either R2 or sign extended bit data from immediate field of instruction = 25 ps
[note - there is no number given for sign extension delay, so we can assume it is less than the register file delay of 150 ps]
5. **ALU** delay: 200 ps

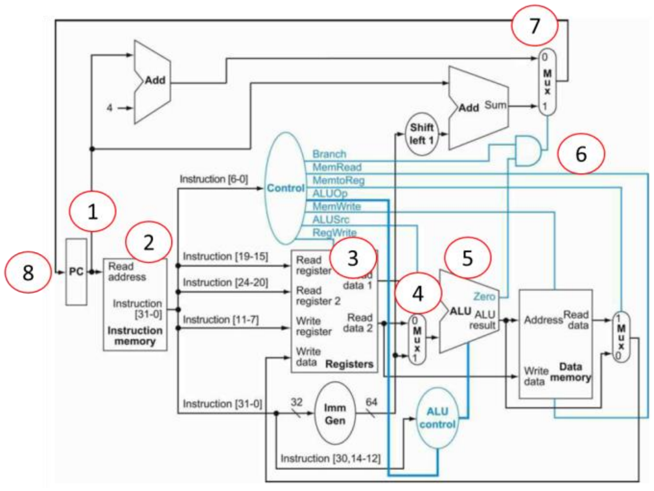
6. **single gate** delay AND of 'zero' output from ALU and 'Branch' control output from Control unit: 5ps

7. **Mux** for Branch address to PC: 25 ps
8. **Write back setup time** *for PC* register: 20 ps

=705ps

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps

beq Instruction Path Delay



2.5 What is the latency of an I-type instruction?
`[same as 2.1 for R-type 'addi'. Latency for ld (problem 2.2) also counts as I-type even though it is different than 'addi']`

2.6 What is the minimum clock period for this CPU?

```
950 ps { longest execution time instruction: ld }
```

3. (a) Suppose you could build a CPU where the clock cycle time was different for each instruction.

3.a1 What would the speedup of this new CPU be over the CPU presented in Figure 4.21 (in RISC-V text) given the instruction mix below?

R-type/I-type (non-ld)	ld	sd	beq
52%	25%	11%	12%

Assuming only the above 4 instructions used with the given frequencies of their occurrence.

cycle time = $T_{R\text{-}type} \times f_{R\text{-}type} + T_{ld} \times f_{ld} + T_{sd} \times f_{sd} + T_{beq} \times f_{beq}$

= 700ps x 0.52 + 950ps x 0.25 + 905ps x 0.11 + 705ps x 0.12 = 785.6ps

Speed-up = 950ps / 785.6 ps = 1.21

(b) Consider the addition of a multiplier to the CPU shown in Figure 4.21. This addition will add 300 ps to the latency of the ALU, but will reduce the number of instructions by 5% (because there will no longer be a need to emulate the multiply instruction).

3.b1 What is the clock cycle time with and without this improvement?

Previous problem: 950ps, with multiplier added: 1250ps

3.b2 What is the speedup achieved by adding this improvement?

time to run new CPU with multiplier reduces by 5%, so (assuming a CPI = 1) execution time from fewer instructions = 0.95 x 1250ps.

Speedup = 950ps / (0.95x 1250ps) = 0.8 (version with multiplier slows down 20%)

3.b3 What is the slowest the new ALU can be and still result in improved performance?

Even if the new CPU has 5% fewer instructions, it can improve the cycle time by at most 50ps. However, this is insufficient to make up for the 300ps penalty by adding a multiplier unit. So, the new ALU cannot be any slower than it is to result in improved performance of the new CPU

(c) When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance trade-off. In the following three problems, assume that we are beginning with the datapath from Figure 4.21, the latencies from Problem 2 in this assignment, and the following costs:

I-Mem	Register File	Mux	ALU	Adder	D-Mem	Single Register	Sign extend	Single gate	Control
1000	200	10	100	30	2000	5	100	1	500

Cycle time from problem 2:

cycle time = $T_{R\text{-}type} x f_{R\text{-}type} + T_{ld} x f_{ld} + T_{sd} x f_{sd} + T_{beq} x f_{beq}$

= 700ps x 0.52 + 950ps x **0.25** + 905ps x **0.11** + 705ps x 0.12 = 785.6ps

Suppose doubling the number of general-purpose registers from 32 to 64 would reduce the number of **ld** and **sd** instruction by 12%, but increase the latency of the register file from 150 ps to 160 ps and double the cost from 200 to 400. (*Use the instruction mix [from 3(a) above] and ignore the other effects on the ISA*)

3.c1 What is the speedup achieved by adding this improvement?

reduction in the number of ld and sd instructions by 12% brings the percentage of load, store instructions down from 36% by 4.2% of all instructions. So cycle time changes from 950ps to (950ps + 10ps) x 0.958 = 919.68ps

speedup = 950/919.68 = 1.03 or 3%

3.c2 Compare the change in performance to the change in cost.

```
Given the number of each component in Fig 4.21 and their costs from the
table given above,
```

Component	**Original CPU**	**New CPU**
PC	5	5
I-Mem	1000	1000
Register File	200	400
ALU	100	100
D-Mem	2000	2000
Sign Extend	100	100
Controls	500	500
Adders	30 x 2	30 x 2
Muxes	10 x 3	10 x 3
Single Gates	1	1
Total	**3996**	**4196**

4196/3996 = 1.05 or increase in cost by 5%

Performance increase of 3% is significant for a cost increase of only 5%

3.c3 Given the cost/performance ratios you just calculated, describe a situation where it makes sense to add more registers and describe a situation where it doesn't make sense to add more registers.

For applications where revenues are very performance sensitive, a mere 3% improvement in performance could easily justify a 10% increase in cost. These examples are generally encountered in HPC processors for the data center. For applications where cost is the primary metric – IoT & embedded systems for example, it would not make sense to make it 10% more expensive to justify a performance improvement of 3%.

4. `ld` is the instruction with the longest latency on the CPU from Section 4.4 (in RISC-V text). If we modified `ld` and sd so that there was no offset (i.e., the address to be loaded from/stored to must be calculated and placed in `rs1` before calling `ld/sd`), then no instruction would use both the ALU and Data memory. This would allow us to reduce the clock cycle time. However, it would also increase the number of instructions, because many `ld` and `sd` instructions would need to be replaced with `ld/add` or `sd/add` combinations.

4.1 What would the new clock cycle time be?

For the new ld instruction , *items 5 and 6 below proceed in parallel*: with the faster of these (ALU delay) removed from the critical path
1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
~~4. Mux to pick data from either R2 or sign extended bit data from immediate field of instruction = 25 ps~~
~~**5. ALU delay: 200 ps**~~
6. Read data from Data Mem, whose address provided by ALU: 250 ps
7. Mux for WB *to Register File* of result from Data Mem: 25 ps
8. Write back setup time *for Register File* registers: 20 ps
----
total ld = 725ps

For the new **sd** instruction , *items 5 and 6 below proceed in parallel*: with the faster of these (ALU delay) removed from the critical path

1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
~~4. Mux to pick data from either R2 or sign extended bit data from immediate field of instruction = 25 ps~~
~~5. ALU delay: 200 ps~~
6. Write Data from R2 to Data Mem: 250 ps

Total for sd: 680 ps

So, new cycle time would be 725ps

4.2 Would a program with the instruction mix presented *in Problem 2* run faster or slower on this new CPU? By how much? (For simplicity, assume every ld and sd instruction is replaced with a sequence of two instructions.)

Since ld/sd instructions are 36% of all instructions in problem 2,3 the number of instructions would increase by 1.36x. So, the new cycle time of 725ps would multiply with 1.36n to give 1020n ps. Speedup = 950 / 986 = 0.963 or a reduction in speed by 3.65%

4.3 What is the primary factor that influences whether a program will run faster or slower on the new CPU?

Since load/store instructions are the slowest instructions, the frequency of their occurrence will impact how much of an improvement the new CPU sees. If the frequency of their occurrence is very small, then the penalty of higher instruction count is small while the improvement in cycle time is much bigger. for example, if only 10% of the program is load/store then then 750ps x 1.1 = 825ps and the speedup is 950/825 = 15%

4.4 Do you consider the original CPU (*as shown in Figure 4.21 of RISC-V text*) a better overall design; or do you consider the new CPU a better overall design? Why?

see above response - depends on how often the ld/sd instruction occur. The improvement to cycle time is 22% by eliminating the ALU op for ld/sd. however, of ld/sd occur more than 22% of instructions then it is not a better CPU

5. (a) Examine the difficulty of adding a proposed `lwi.d rd, rs1, rs2` ("Load With Increment") instruction to RISC-V. Interpretation: `Reg[rd]=Mem[Reg[rs1]+Reg[rs2]]`

Most RISC type ISAs use a load-store model for memory access which means that only load/store (lw and sw) instructions can access memory. While on x86 most instructions are allowed to directly operate on data in memory, in MIPS or RISCV, for example, data must be moved from memory into registers before being operated on. *This means that incrementing a 32-bit value at a particular memory address* in MIPS/RISCV would require three types of instructions (load, increment, and store) to first load the value at a particular address into a register, increment it within the register, and store it back to the memory from the register. This offset form uses a register as an offset. An example usage of this offset form is when the code wants to access an array where the index is computed at run-time

The proposed LWI.d instruction would perform 2 of the above three tasks in one instruction – *load with increment* where the content of register rs1 which holds a memory address is added to register rs2 (with the data held at the memory address corresponding to the sum of these 2 registers requested by LWI) . The load operation requests the content in memory corresponding to this summed address be brought in to register rd.

The suffix .d to lwi.d corresponds to the shifter that is used to scale the register offset in rs2.

5.a1 Which new functional blocks (if any) do we need for this instruction?

The ALU could be used to add the 2 registers rs1 and rs2, the shifter could be used to shift the offset in register rs2, so no new hardware is necessary

5.a2 Which existing functional blocks (if any) require modification?

Only the control unit needs modification

5.***a3*** Which new data paths (if any) do we need for this instruction?

No new data paths are needed.

6. In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

IF	**ID**	**EX**	**MEM**	**WB**
250 ps	350 ps	150 ps	300 ps	200 ps

Also, assume that instructions executed by the processor are broken down as follows:

ALU/Logic	**Jump/Branch**	**Load**	**Store**
45%	20%	20%	15%

6.1 What is the clock cycle time in a pipelined and non-pipelined processor?
pipelined 350ps (slowest stage) non-pipelined 1250ps

6.2 What is the total latency of an `ld` instruction in a pipelined and non-pipelined processor?
pipelined and non-pipelined = 1250ps

6.3 If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?
The slowest stage: 350ps. Now pipelined version is 300ps cycle time

6.4 Assuming there are no stalls or hazards, what is the utilization of the data memory?
load and store = 35% (the only instructions that access data memory)

6.5 Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?

65% - only ALU and load utilize write back to register file write port

7. What is the minimum number of cycles needed to completely execute n instructions on a CPU with a k stage pipeline? Justify your formula.

```
For a k stage pipeline, the first instruction reaches the end of
the pipeline in k cycles.

After this, the remaining n-1 instructions execute at 1 cycle
per instruction over the next n-1 cycles

so, n instructions take k + n -1 cycles to execute in a k stage
pipeline
```

8. (a) Assume that `x11` is initialized to `11` and `x12` is initialized to `22`. Suppose you executed the code below on a version of the pipeline from Section 4.5 that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). What would the final values of registers `x13` and `x14` be?

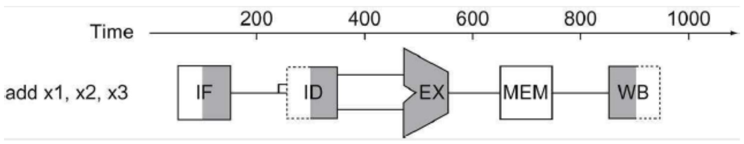
```
addix11, x12, 5

addx13, x11, x12

addix14, x11, 15
```

`x13 = 33, x14 = 26`

(b) Assume that `x11` is initialized to `11` and `x12` is initialized to `22`. Suppose you executed the code below on a version of the pipeline from Section 4.5 *that does not handle data hazards* (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary).

What would the final values of register `x15` be? Assume the register file is written at the beginning of the cycle and read at the end of a cycle. Therefore, an `ID` stage will return the results of a `WB` state occurring during the same cycle. See Section 4.7 and Figure 4.51 for details.

```
addix11, x12, 5
addx13, x11, x12
addix14, x11, 15
addx15, x11, x11
```



```
addix11, x12, 5         #x11 does not update to 27 (=22+5) until the add
                        instruction below

addx13, x11, x12        #x13 = 33

addix14, x11, 15        #x11 = 26

addx15, x11, x11        #x15 = 54
```