UNIVERSITI TEKNOLOGI PETRONAS

FINAL EXAMINATION
JANUARY 2025 SEMESTER

COURSE : EDB4033 - COMPUTER SYSTEMS ARCHITECTURE

DATE : 14 APRIL 2025 (MONDAY)

TIME : 2.30 PM - 5.30 PM (3 HOURS)

INSTRUCTIONS TO CANDIDATES

1. Answer **ALL** questions in the Answer Booklet.
2. Begin **EACH** answer on a new page in the Answer Booklet.
3. Indicate clearly answers that are cancelled, if any.
4. Where applicable, show clearly steps taken in arriving at the solutions and indicate **ALL** assumptions, if any.
5. **DO NOT** open this Question Booklet until instructed.

Note :

i. There are **FOURTEEN (14)** pages in this Question Booklet including the cover page and appendices.

ii. **DOUBLE-SIDED** Question Booklet.

1. a. A 4-bit ripple-carry adder is to be built using the 1-bit full adder design in **FIGURE Q1**. All gates have the following propagation delays; inverters have a delay of 2 ns, 2-input AND and OR gates have a 6 ns delay, 2-input NAND and NOR gates have 8 ns delay and 3-input OR gates have a delay of 4 ns.

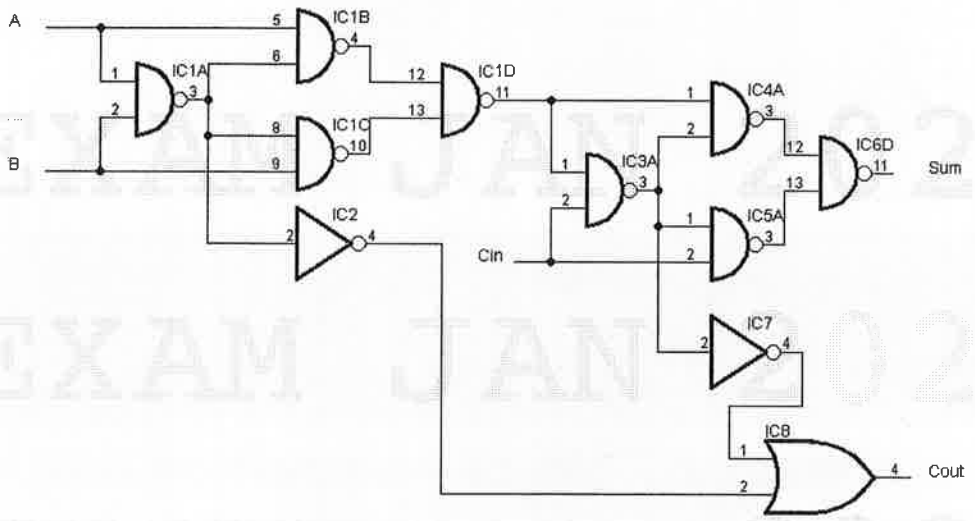

FIGURE Q1

Evaluate the worst case path or critical path for the 4-bit adder designed using the 1-bit adder design illustrated in **FIGURE Q1**.

[8 marks]

b. Based on the RISC-V instruction set architecture and the numbered registers available in a RISC-V processor, determine if the following instructions are able to be compiled and/or able to be executed in a correct manner:

i. `beq x1 , x2, 0x1ffff`

[4 marks]

ii. `jalr 0x3ffffff`

[4 marks]

2. The single cycle RISC-V datapath illustrated in **FIGURE Q2** is capable of only running some instructions.

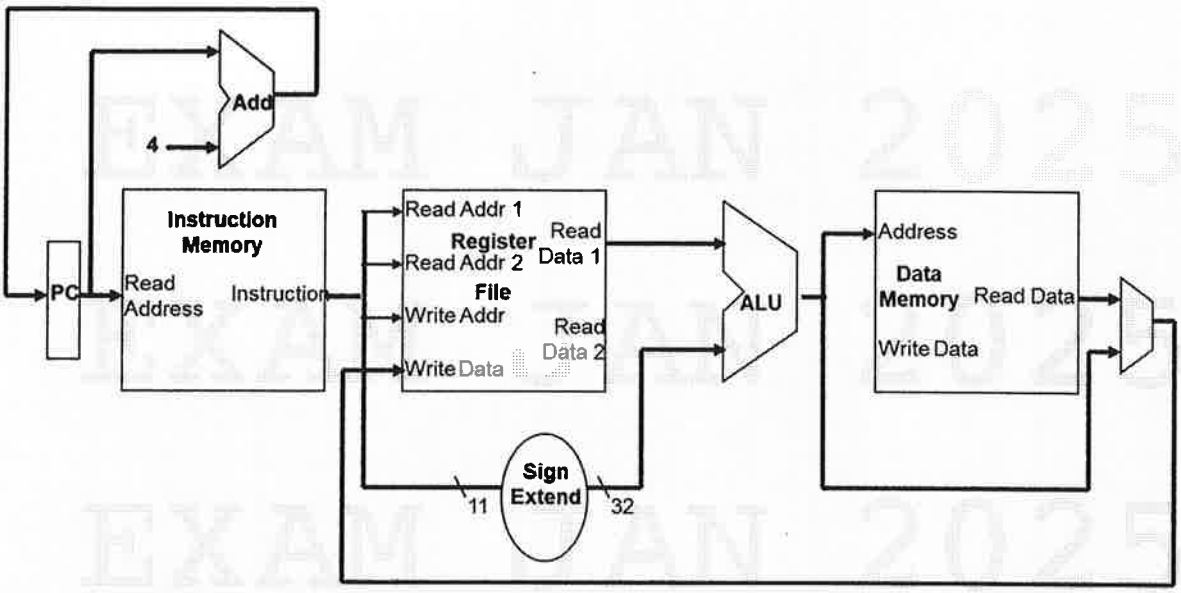

FIGURE Q2

a. Suggest and speculate the possible type of instructions or class of instructions that can be executed on the datapath illustrated in **FIGURE Q2**.

[8 marks]

b. A new instruction as listed in **TABLE Q2b** is to be added to the datapath illustrated in **FIGURE Q2**. Design and modify the current datapath in **FIGURE Q2** to be able to execute the following instructions **in TABLE Q2b**. Also determine the control signals needed for all instructions to be executed on this data path.

TABLE Q2b

Instruction	Instruction Operation
and rd,rs1, rs2	Reg[rd] ← Reg[rs1] + Reg[rs2]
lw rd , imm (rs1)	Reg[rd] <- Mem (Reg[rs1] + SignEx [Imm])
sw rt, Imm(rs1)	Mem (Reg[rs1] + SignEx [Imm]) <- Reg[rt]

[15 marks]

c. Based on the datapath in **FIGURE Q2**, the following functional units have latencies as listed in **TABLE Q2c**.

TABLE Q2c

Functional Unit	Latency (ps)
Instruction Memory	350
Adder	100
Multiplexer	60
ALU	250
Register File	200
Data Memory	550
Program Counter	100
Sign Extender	100

List all possible critical paths in the datapath. Determine the clock cycle time required for the single cycle processor.

[5 marks]

3. In a 5-stage pipelined architecture, the following piece of code is executed:

```
lw   x5, 0(x1)
lw   x3, 8(x5)
add  x6, x5, x3
lw   x4, 8(x6)
add  x6, x6, x4
sub  x7, x4, x6
sw   x6, 8(x7)
```

a. Analyze the above code and discuss if there are dependencies and type of data hazards.

[8 marks]

b. Assume that the CPU is now equipped with a Hazard Detection Unit that is able to insert bubbles and stall the pipeline. Using a clock cycle table, analyze the execution of the program as it passes through the pipeline stages. Use the following standardized notation to indicate the pipeline stage for each instruction:

IF - Instruction fetch
ID - Instruction Decode
EX - Execution
MEM - Memory Access
WB - Write back

[8 marks]

c. To further enhance the CPU, a forwarding unit was added to the Hazard Detection unit of **part (b)**. Using a clock cycle table, show the execution of the program as it passes through the pipeline stages. Evaluate the approach used for code execution using data forwarding and bubbles, and use arrows to indicate the forwarded registers and the stages where the data is being forwarded from the source stage to the destination stage. Use the same notation as in **part (b)**.

[10 marks]

4. In a computer memory system, a series of address references are applied as hexadecimal word addresses: 0x10, 0x12, 0x13, 0x14, 0x15, 0x17, 0x19, 0x1a, 0x16, 0x14, 0x13, 0x15, 0x17, 0x1c, 0x16, 0x13.

a. Assuming that a direct mapped cache with 2 line capacity and one-word block is initially empty, label each reference in the list as a hit, compulsory miss or conflict miss and show the final contents of the cache and tags in the table in **APPENDIX I**.

[10 marks]

b. Assume that a four-way set-associative cache, with 2-line capacity and two-word blocks is initially empty. Label each reference in the list as a hit, compulsory miss or conflict miss and show the final contents of the cache and tags in the table in **APPENDIX II**.

[15 marks]

c. Determine the hit rate for each cache in **part (a)** and **part (b),** and speculate on the cache parameters that would affect the hit rate in the cache and the limitation of these changes.

[5 marks]

-END OF PAPER-

APPENDIX I

Question 4a :

Detach this **APPENDIX I** and attach it to your answer booklet

Examination ID : _____

Hit / Miss Identification

Address Reference	Hit / Compulsory Miss / Conflict Miss
0x10	
0x12	
0x13	
0x14	
0x15	
0x17	
0x19	
0x1a	
0x16	
0x14	
0x13	
0x15	
0x17	
0x1c	
0x16	
0x13	

APPENDIX I (continued)

Final State of Cache

Block Number	Tag	Word 0 Data
0		
1		

APPENDIX II

Question 4b :

Detach this **APPENDIX II** and attach it to your answer booklet

Examination ID : _____

Hit / Miss Identification

Address Reference	Hit / Compulsory Miss / Conflict Miss
0x10	
0x12	
0x13	
0x14	
0x15	
0x17	
0x19	
0x1a	
0x16	
0x14	
0x13	
0x15	
0x17	
0x1c	
0x16	
0x13	

APPENDIX II (continued)

Final State of Cache

Block Number	Set #0			Set #1		
	Tag	Word 0	Word 1	Tag	Word0	Word 1
0						
1						

Block Number	Set #2			Set #3		
	Tag	Word 0	Word 1	Tag	Word0	Word 1
0						
1						

APPENDIX III

RISC-V Green Sheet

MNEMONIC	FMT	NAME	DESCRIPTION (in Verilog)	N
add, addw	R	ADD (Word)	R[rd] = R[rs1] + R[rs2]	
addi, addiw	I	ADD Immediate (Word)	R[rd] = R[rs1] + imm	
and	R	AND	R[rd] = R[rs1] & R[rs2]	
andi	I	AND Immediate	R[rd] = R[rs1] & imm	
auipc	U	Add Upper Immediate to PC	R[rd] = PC + {imm, 12'b0}	
beq	SB	Branch EQual	if(R[rs1]==R[rs2]) PC=PC + {imm,1b'0}	
bge	SB	Branch Greater than or Equal	if(R[rs1]>=R[rs2]) PC=PC + {imm,1b'0}	
bgeu	SB	Branch ≥ Unsigned	if(R[rs1]>=R[rs2]) PC=PC + {imm,1b'0}	
blt	SB	Branch Less Than	if(R[rs1]<R[rs2]) PC=PC+ {imm,1b'0}	
bltu	SB	Branch Less Than Unsigned	if(R[rs1]<R[rs2]) PC=PC + {imm,1b'0}	
bne	SB	Branch Not Equal	if(R[rs1]!=R[rs2]) PC=PC + {imm,1b'0}	
csrrc	I	Cont./Stat.RegRead&Clear	R[rd] = CSR;CSR = CSR & ~R[rs1]	
csrrci	I	Cont./Stat.RegRead&Clear Imm	R[rd] = CSR;CSR = CSR & ~imm	
csrrs	I	Cont./Stat.RegRead&Set	R[rd] = CSR; CSR = CSR | R[rs1]	
csrrsi	I	Cont./Stat.RegRead&Set Imm	R[rd] = CSR; CSR = CSR | imm	
csrrw	I	Cont./Stat.RegRead&Write	R[rd] = CSR; CSR = R[rs1]	
csrrwi	I	Cont./Stat.Reg Read&Write Imm	R[rd] = CSR; CSR = imm	
ebreak	I	Environment BREAK	Transfer control to debugger	
ecall	I	Environment CALL	Transfer control to operating system	
fence	I	Synch thread	Synchronizes threads	
fence.i	I	Synch Instr & Data	Synchronizes writes to instruction stream	
jal	UJ	Jump & Link	R[rd] = PC+4; PC = PC + {imm,1b'0}	
jalr	I	Jump & Link Register	R[rd] = PC+4; PC = R[rs1]+imm	
lb	I	Load Byte	R[rd] = {56'bM[](7),M[R[rs1]+imm](7:0)}	
lbu	I	Load Byte Unsigned	R[rd] = {56'b0,M[R[rs1]+imm](7:0)}	
ld	I	Load Doubleword	R[rd] = M[R[rs1]+imm](63:0)	
lh	I	Load Halfword	R[rd] = {48'bM[](15),M[R[rs1]+imm](15:0)}	
lhu	I	Load Halfword Unsigned	R[rd] = {48'b0,M[R[rs1]+imm](15:0)}	
lui	U	Load Upper Immediate	R[rd] = {32b'imm<31>, imm, 12'b0}	
lw	I	Load Word	R[rd] = {32'bM[](31),M[R[rs1]+imm](31:0)}	
lwu	I	Load Word Unsigned	R[rd] = {32'b0,M[R[rs1]+imm](31:0)}	
or	R	OR	R[rd] = R[rs1] | R[rs2]	
ori	I	OR Immediate	R[rd] = R[rs1] | imm	
sb	S	Store Byte	M[R[rs1]+imm](7:0) = R[rs2](7:0)	
sd	S	Store Doubleword	M[R[rs1]+imm](63:0) = R[rs2](63:0)	
sh	S	Store Halfword	M[R[rs1]+imm](15:0) = R[rs2](15:0)	
sll, sllw	R	Shift Left (Word)	R[rd] = R[rs1] << R[rs2]	
slli, slliw	I	Shift Left Immediate (Word)	R[rd] = R[rs1] << imm	
slt	R	Set Less Than	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0	
slti	I	Set Less Than Immediate	R[rd] = (R[rs1] < imm) ? 1 : 0	
sltiu	I	Set < Immediate Unsigned	R[rd] = (R[rs1] < imm) ? 1 : 0	
sltu	R	Set Less Than Unsigned	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0	
sra, sraw	R	Shift Right Arithmetic (Word)	R[rd] = R[rs1] >> R[rs2]	
srai, sraiw	I	Shift Right Arith Imm (Word)	R[rd] = R[rs1] >> imm	
srl, srlw	R	Shift Right (Word)	R[rd] = R[rs1] >> R[rs2]	
srli, srliw	I	Shift Right Immediate (Word)	R[rd] = R[rs1] >> imm	
sub, subw	R	SUBtract (Word)	R[rd] = R[rs1] - R[rs2]	
sw	S	Store Word	M[R[rs1]+imm](31:0) = R[rs2](31:0)	
xor	R	XOR	R[rd] = R[rs1] ^ R[rs2]	
xori	I	XOR Immediate	R[rd] = R[rs1] ^ imm	

OPCODES IN NUMERICAL ORDER BY OPCODE

MNEMONIC	FMT	OPCODE	FUNCT3	FUNCT7 OR IMM	HEXADECIM
lb	I	0000011	000		03/0
lh	I	0000011	001		03/1
lw	I	0000011	010		03/2
ld	I	0000011	011		03/3
lbu	I	0000011	100		03/4
lhu	I	0000011	101		03/5
lwu	I	0000011	110		03/6
fence	I	0001111	000		0F/0
fence.i	I	0001111	001		0F/1
addi	I	0010011	000		13/0
slli	I	0010011	001	0000000	13/1/00
slti	I	0010011	010		13/2
sltiu	I	0010011	011		13/3
xori	I	0010011	100		13/4
srli	I	0010011	101	0000000	13/5/00
srai	I	0010011	101	0100000	13/5/20
ori	I	0010011	110		13/6
andi	I	0010011	111		13/7
auipc	U	0010111			17
addiw	I	0011011	000		1B/0
slliw	I	0011011	001	0000000	1B/1/00
srliw	I	0011011	101	0000000	1B/5/00
sraiw	I	0011011	101	0100000	1B/5/20
sb	S	0100011	000		23/0
sh	S	0100011	001		23/1
sw	S	0100011	010		23/2
sd	S	0100011	011		23/3
add	R	0110011	000	0000000	33/0/00
sub	R	0110011	000	0100000	33/0/20
sll	R	0110011	001	0000000	33/1/00
slt	R	0110011	010	0000000	33/2/00
sltu	R	0110011	011	0000000	33/3/00
xor	R	0110011	100	0000000	33/4/00
srl	R	0110011	101	0000000	33/5/00
sra	R	0110011	101	0100000	33/5/20
or	R	0110011	110	0000000	33/6/00
and	R	0110011	111	0000000	33/7/00
lui	U	0110111			37
addw	R	0111011	000	0000000	3B/0/00
subw	R	0111011	000	0100000	3B/0/20
sllw	R	0111011	001	0000000	3B/1/00
srlw	R	0111011	101	0000000	3B/5/00
sraw	R	0111011	101	0100000	3B/5/20
beq	SB	1100011	000		63/0
bne	SB	1100011	001		63/1
blt	SB	1100011	100		63/4
bge	SB	1100011	101		63/5
bltu	SB	1100011	110		63/6
bgeu	SB	1100011	111		63/7
jalr	I	1100111	000		67/0
jal	UJ	1101111			6F
ecall	I	1110011	000	000000000000	73/0/000
ebreak	I	1110011	000	000000000001	73/0/001
csrrw	I	1110011	001		73/1
csrrs	I	1110011	010		73/2
csrrc	I	1110011	011		73/3
csrrwi	I	1110011	101		73/5
csrrsi	I	1110011	110		73/6
csrrci	I	1110011	111		73/7

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

REGISTER NAME, USE, CALLING CONVENTION

④

REGISTER	NAME	USE	SAVER
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/Frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function arguments/Return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller
f0-f7	ft0-ft7	FP Temporaries	Caller
f8-f9	fs0-fs1	FP Saved registers	Callee
f10-f11	fa0-fa1	FP Function arguments/Return values	Caller
f12-f17	fa2-fa7	FP Function arguments	Caller
f18-f27	fs2-fs11	FP Saved registers	Callee
f28-f31	ft8-ft11	R[rd] = R[rs1] + R[rs2]	Caller