Princess Sumaya University for Technology
جامعة الأميرة سمية للتكنولوجيا

Dept. of Computer Engineering
22541 Computer Architecture
Spring 2022- Final Exam

Name (In Arabic): Solution
Student ID: .....
Instructors: Dr. Awos Kanan
Duration: 120 minutes.
Date: 15/6/2022

Instructions:.
- *Exam consists of Six questions, with total of 40 points.*
- *Show your work, final answer by itself does not count.*
- *RISC-V Instruction Set Sheet is provided in the last page*
- *Cell phones are not allowed.*

Q 1	/15	Q 4	/6
Q 2	/4	Q 5	9/
Q 3	/3	Q 6	3/

Total Grade: /40

Question 1: Multiple Choice

(15 points)

1. What is the fastest storage unit in a typical memory hierarchy?

a. Main memory.
b. Cache memory
c. Hard Disk.
d. Flash memory.

2. The method for updating the main memory as soon as a block is removed from the cache is called:

a. Write Back.
b. Write Through.
c. Write Allocate.
d. Write Around.

3. A major advantage of direct mapping of a cache is its simplicity. The main disadvantage of this organization is that:

a. It does not allow simultaneous access to the intended data and its tag.
b. It is more expensive than other types of cache organizations.
c. It does not utilize temporal locality of programs.
d. It does not utilize spatial locality of programs.

4. The write method that is used to update the disk when a page in main memory is updated is:

a. Write Back
b. Write Through
c. Write Allocate
d. Write Around

5. If a cache access requires one clock cycle, and assuming that handling 'cache misses' stalls the processor for an additional five cycles, which of the following cache hit rates results in achieving an average memory access of 2 cycles?

a. 75%
b. 80%
c. 83%
d. 86%

6. What must be considered when designing a cache system?

a. Number and types of caches.
b. Mapping and replacement algorithms.
c. Write policies.
d. All of the above.

7. SIMD processors improves performance by

a. Decreasing instruction latency.
b. Eliminating data hazards.
c. Exploiting instruction-level parallelism.
d. Exploiting data-level parallelism.

8. What is the speedup of a parallel program that achieves an efficiency of 80% on 50 processors?

a. 1.6
b. 40
c. 16
d. None of the above

9. In K-stage pipelined processor, we are able to start executing a new instruction at each clock cycle. However, this requires using pipeline register to hold the data between cycles. How many pipeline registers are needed?

a. K registers having fixed sizes.
b. (K-1) registers having variable sizes.
c. (K-1) registers having fixed sizes.
d. K registers having variable sizes.

10. In a pipelined RISC-V architecture, state how the control signals are delivered to the different pipeline stages:

a. The pipelines registers are extended to pass the control signals.
b. Each stage has a sub-control unit connected to a centralized control unit.
c. Each pipeline stage is directly connected to a centralized control unit.
d. The pipeline stages are self-controlled: no control signals are needed.

11. What effect do pipeline stalls (or bubbles) have on CPI?

a. Have no effect on CPI.
b. Decrease CPI.
c. Increase CPI.
d. May increase or decrease CPI depending on the type if instructions.

12. Forwarding circuits are added to the RISC-V pipelined architecture to:

a. Resolve Structural Hazards.
b. Resolve Data Hazards.
c. Resolve Control hazards.
d. Resolve all types of hazards.

13. If a CPU completes the execution of 100 instructions in 10,000 ns, then the processor has a MIPS (Million Instructions Per Second) performance equal to:

a. 10
b. 10×10^3
c. 100
d. 10×10^6

14. Translate the C code (x = y + 2*z;) to RISC-V. Assume that the variables x, y and z are already stored in registers X0, X1 and X2 respectively.

a.
```
sll X1,X2,1
add X0,X1,X2
```

b.
```
sll X2,X2,1
add  X0,X1,X2
```

c.
```
sll X2,X2,2
add X0,X1,X2
```

d.
```
add X0,X1,X2
```

15. The main advantage of using a Translation Look-aside Buffer (TLB) is:

a. A TLB ensures that a process does not access memory outside of its address space.
b. A TLB makes translating virtual addresses to physical addresses faster.
c. A TLB allows multiple processes to share the L1 cache.
d. A TLB makes translating virtual addresses to physical addresses possible.

Question 2: Performance

(4 points)

a. Assume a program takes 10.7 billion instructions to execute on a processor and the processor has an average CPI of 1.5 for that program. What would the clock rate of the processor need to be to allow the program to execute in two seconds? **(2 points)**

$$\text{CPU Time} = \frac{IC * CPI}{f}$$

$$\Rightarrow f = \frac{10.7 * 10^9 * 1.5}{2} = 8.025 \text{ GHz}$$

b. A proposed hardware optimization for a given processor would eliminate 10% of instructions and decrease the CPI of the remaining instructions by 10%. However, this optimization would also result in decreasing the clock rate by 14%. Is this optimization worth implementing? Show your calculations. **(2 points)**

$$\text{CPU Time}_{org.} = \frac{IC * CPI}{f}$$

$$\text{CPU Time}_{new} = \frac{0.9\,IC * 0.9\,CPI}{0.86\,f} = 0.94\ \text{CPU Time}_{org.}$$

Less by 6% —— Yes, worth implementing

Question 3: Instruction Set Architecture

(3 points)

Write a minimum piece of RISC-V assembly code to implement the following C function:
Hint: No need to push/pop any variable to/from the stack.

```
int F (int A, int B)
{
if (A > B)
return A;
else
return B;
}
```

Assume
return address in X_1
A in X_{11}
B in X_{12}
Return value in X_{13}

```
F:    bge  x12, x11, else
      addi x13, x11, 0
      beq  x0, x0, exit
else: addi x13, x10, 0
exit: jalr x0, 0(x1)
```

Question 4: Pipelining

(6 points)

For the following piece of RISC-V assembly code:

```
add  X1, X2, X3
addi X4, X2, 1
or   X5, X1, X3
ld   X6, 0(X1)
nor  X7, X6, X1
```

and the following cycle times:
Single-Cycle (Non-Pipelined): 200 ps
5-Stage Pipelined (No Forwarding): 50 ps
5-Stage Pipelined (Full Forwarding): 80 ps

Hint: For partial credits, show your steps

What is the execution time of this code on:

- Single Cycle Implementation:

$5 \times 200 = 1000$ ps

- 5-Stage Pipelined Implementation (No Forwarding)

$(5+8-1) * 50 = 600$ ps

(3 stalls)

- 5-Stage Pipelined Implementation (Full Forwarding)

$(5+6-1) * 80 = 800$ ps

(1 stall)

Question 5: Memory Hierarchy **(9 points)**

a) For a direct-mapped cache with physical address divided as follows: **(6 points)**

2-bit	13-bit	5-bit
Tag	**Index**	**Byte Offset**

1- What is the maximum size of the main memory?

$2^{20} = 1$ MB

2- What is the block size?

$2^5 = 32$ Bytes

3- How many blocks there are in this cache memory?

$2^{13} = 8$ K blocks

4- What is the size of this cache memory?

$2^{13} * 2^5 = 2^{18} = 256$ KB

5- What is the associativity of this cache memory? (N-way, N=?)

1 - way

6- How would you enhance the utilization of spatial locality for this cache memory?

Increase block size

b) If the cache memory in <u>part a</u> is <u>upgraded to become a 4-way set associative cache (with the same breakdown of address fields above)</u>: (3 points)

1- What is the size of the upgraded cache memory?

$4 * 256\ KB = 2^{20} = 1\ MB$

2-Explain briefly how would this upgrade reduce cache miss rate?

Enhance utilization of temporal locality

3- How many comparators are needed for this cache? What is the width (in bits) of these comparators?

4 comparators, 2-bit each

<u>Question 6: Parallel Processing</u> (3 points)

Assume that a single processor takes T seconds to multiply two numbers. Consider a program that consists of 50 multiplications that can't be parallelized and 500 multiplications that can be executed in parallel.

a) What is the <u>speedup</u> and <u>efficiency</u> for running this program on 128 processors?

$T_{seq.} = (50 + 500) * T = 550\,T$

$T_{parallel} = 50T + \frac{500}{128} T = 53.9\,T$

$Speedup = \frac{550T}{53.9T} = 10.2$

$Efficiency = \frac{10.2}{128} = 0.08 \approx 8\%$

b) How many steps are required by these processors to perform a <u>reduction</u> operation?

$\log_2 128 = 7$ Steps

RISC-V Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

RISC-V Reference Data

①

RV32I BASE INTEGER INSTRUCTIONS, in alphabetical order

MNEMONIC	FMT	NAME	DESCRIPTION (in Verilog)	NOTE
add	R	ADD	R[rd] = R[rs1] + R[rs2]	
addi	I	ADD Immediate	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}	2)
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}	2)
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] = {24'bM[](7),M[R[rs1]+imm](7:0)}	3) 4)
lbu	I	Load Byte Unsigned	R[rd] = {24'b0,M[R[rs1]+imm](7:0)}	
lh	I	Load Halfword	R[rd] = {16'bM[](15),M[R[rs1]+imm](15:0)}	
lhu	I	Load Halfword Unsigned	R[rd] = {16'b0,M[R[rs1]+imm](15:0)}	4)
lui	U	Load Upper Immediate	R[rd] = {imm, 12'b0}	
lw	I	Load Word	R[rd] = {M[R[rs1]+imm](31:0)}	
or	R	OR	R[rd] = R[rs1] | R[rs2]	
ori	I	OR Immediate	R[rd] = R[rs1] | imm	4)
sb	S	Store Byte	M[R[rs1]+imm](7:0) = R[rs2](7:0)	
sh	S	Store Halfword	M[R[rs1]+imm](15:0) = R[rs2](15:0)	
sll	R	Shift Left	R[rd] = R[rs1] << R[rs2]	
slli	I	Shift Left Immediate	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	R	Shift Right Arithmetic	R[rd] = R[rs1] >> R[rs2]	
srai	I	Shift Right Arith Imm	R[rd] = R[rs1] >> imm	2)
srl	R	Shift Right (Word)	R[rd] = R[rs1] >> R[rs2]	2)
srli	I	Shift Right Immediate	R[rd] = R[rs1] >> imm	5)
sub, subw	R	SUBtract (Word)	R[rd] = R[rs1] – R[rs2]	5)
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	

Notes: 1) Operation assumes unsigned integers (instead of 2's complement)
2) The least significant bit of the branch address in jalr is set to 0
3) (signed) Load instructions extend the sign bit of data to fill the 32-bit register
4) Replicates the sign bit to fill in the leftmost bits of the result during right shift
5) Multiply with one operand signed and one unsigned
6) The Single version does a single-precision operation using the rightmost 32 bits of a 64-bit F register
7) Classify writes a 10-bit mask to show which properties are true (e.g., –inf, -0,+0, +inf, denorm, ...)
8) Atomic memory operation; nothing else can interpose itself between the read and the write of the memory location
The immediate field is sign-extended in RISC-V

ARITHMETIC CORE INSTRUCTION SET

②

RV64M Multiply Extension

MNEMONIC	FMT	NAME	DESCRIPTION (in Verilog)	NOTE
mul	R	MULtiply	R[rd] = (R[rs1] * R[rs2])(63:0)	
mulh	R	MULtiply High	R[rd] = (R[rs1] * R[rs2])(127:64)	
mulhsu	R	MULtiply High Unsigned	R[rd] = (R[rs1] * R[rs2])(127:64)	2)
mulhu	R	MULtiply upper Half Unsigned	R[rd] = (R[rs1] * R[rs2])(127:64)	6)
div	R	DIVide	R[rd] = (R[rs1] / R[rs2])	
divu	R	DIVide Unsigned	R[rd] = (R[rs1] / R[rs2])	2)
rem	R	REMainder	R[rd] = (R[rs1] % R[rs2])	
remu	R	REMainder Unsigned	R[rd] = (R[rs1] % R[rs2])	2)

RV64F and RV64D Floating-Point Extensions

fld, flw	I	Load (Word)	F[rd] = M[R[rs1]+imm]	
fsd, fsw	S	Store (Word)	M[R[rs1]+imm] = F[rd]	
fadd.s, fadd.d	R	ADD	F[rd] = F[rs1] + F[rs2]	7)
fsub.s, fsub.d	R	SUBtract	F[rd] = F[rs1] – F[rs2]	7)
fmul.s, fmul.d	R	MULtiply	F[rd] = F[rs1] * F[rs2]	7)
fdiv.s, fdiv.d	R	DIVide	F[rd] = F[rs1] / F[rs2]	7)
fsqrt.s, fsqrt.d	R	SQuare RooT	F[rd] = sqrt(F[rs1])	7)
fmadd.s, fmadd.d	R	Multiply-ADD	F[rd] = F[rs1] * F[rs2] + F[rs3]	7)
fmsub.s, fmsub.d	R	Multiply-SUBtract	F[rd] = F[rs1] * F[rs2] - F[rs3]	7)
fmnsub.s, fmnsub.d	R	Negative Multiply-ADD	F[rd] = -(F[rs1] * F[rs2] – F[rs3])	7)
fmnadd.s, fmnadd.d	R	Negative Multiply-SUBtract	F[rd] = -(F[rs1] * F[rs2] + F[rs3])	7)
fsgnj.s, fsgnj.d	R	SiGN source	F[rd] = { F[rs2]<63>,F[rs1]<62:0>}	7)
fsgnjn.s, fsgnjn.d	R	Negative SiGN source	F[rd] = { (~F[rs2]<63>), F[rs1]<62:0>}	7)
fsgnjx.s, fsgnjx.d	R	Xor SiGN source	F[rd] = {F[rs2]<63>^F[rs1]<63>, F[rs1]<62:0>}	7)
fmin.s, fmin.d	R	MINimum	F[rd] = (F[rs1] < F[rs2]) ? F[rs1] : F[rs2]	7)
fmax.s, fmax.d	R	MAXimum	F[rd] = (F[rs1] > F[rs2]) ? F[rs1] : F[rs2]	7)
feq.s, feq.d	R	Compare Float EQual	R[rd] = (F[rs1]== F[rs2]) ? 1 : 0	7)
flt.s, flt.d	R	Compare Float Less Than	R[rd] = (F[rs1]< F[rs2]) ? 1 : 0	7)
fle.s, fle.d	R	Compare Float Less than or =	R[rd] = (F[rs1]<= F[rs2]) ? 1 : 0	7)
fclass.s, fclass.d	R	Classify Type	R[rd] = class(F[rs1])	7,8)
fmv.s.x, fmv.d.x	R	Move from Integer	F[rd] = R[rs1]	7)
fmv.x.s, fmv.x.d	R	Move to Integer	R[rd] = F[rs1]	7)
fcvt.d.s	R	Convert from SP to DP	F[rd] = single(F[rs1])	
fcvt.s.d	R	Convert from DP to SP	F[rd] = double(F[rs1])	
fcvt.s.w, fcvt.d.w	R	Convert from 32b Integer	F[rd] = float(R[rs1](31:0))	7)
fcvt.s.l, fcvt.d.l	R	Convert from 64b Integer	F[rd] = float(R[rs1](63:0))	7)
fcvt.s.wu, fcvt.d.wu	R	Convert from 32b Int Unsigned	F[rd] = float(R[rs1](31:0))	2,7)
fcvt.s.lu, fcvt.d.lu	R	Convert from 64b Int Unsigned	F[rd] = float(R[rs1](63:0))	2,7)
fcvt.w.s, fcvt.w.d	R	Convert to 32b Integer	R[rd](31:0) = integer(F[rs1])	7)
fcvt.l.s, fcvt.l.d	R	Convert to 64b Integer	R[rd](63:0) = integer(F[rs1])	7)
fcvt.wu.s, fcvt.wu.d	R	Convert to 32b Int Unsigned	R[rd](31:0) = integer(F[rs1])	2,7)
fcvt.lu.s, fcvt.lu.d	R	Convert to 64b Int Unsigned	R[rd](63:0) = integer(F[rs1])	2,7)

RV64A Atomic Extension

amoadd.w, amoadd.d	R	ADD	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] + R[rs2]	9)
amoand.w, amoand.d	R	AND	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] & R[rs2]	9)
amomax.w, amomax.d	R	MAXimum	R[rd] = M[R[rs1]], if (R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]	9)
amomaxu.w, amomaxu.d	R	MAXimum Unsigned	R[rd] = M[R[rs1]], if (R[rs2] > M[R[rs1]]) M[R[rs1]] = R[rs2]	2,9)
amomin.w, amomin.d	R	MINimum	R[rd] = M[R[rs1]], if (R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]	9)
amominu.w, amominu.d	R	MINimum Unsigned	R[rd] = M[R[rs1]], if (R[rs2] < M[R[rs1]]) M[R[rs1]] = R[rs2]	2,9)
amoor.w, amoor.d	R	OR	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] | R[rs2]	9)
amoswap.w, amoswap.d	R	SWAP	R[rd] = M[R[rs1]], M[R[rs1]] = R[rs2]	9)
amoxor.w, amoxor.d	R	XOR	R[rd] = M[R[rs1]], M[R[rs1]] = M[R[rs1]] ^ R[rs2]	9)
lr.w, lr.d	R	Load Reserved	R[rd] = M[R[rs1]], reservation on M[R[rs1]]	
sc.w, sc.d	R	Store Conditional	if reserved, M[R[rs1]] = R[rs2], R[rd] = 0; else R[rd] = 1	

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

© 2021 by Elsevier, Inc. All rights reserved. From Patterson and Hennessy, *Computer Organization and Design: The Hardware/Software Interface: RISC-V Edition*