CS2100

28 Feb 2025

Revision #1

(Prepared by: Aaron Tan)

Contents:

Important:

- Date: 12 March 2025, Wednesday
- Time: 7 – 8:30pm
- Venue: MPSH2B

Format:

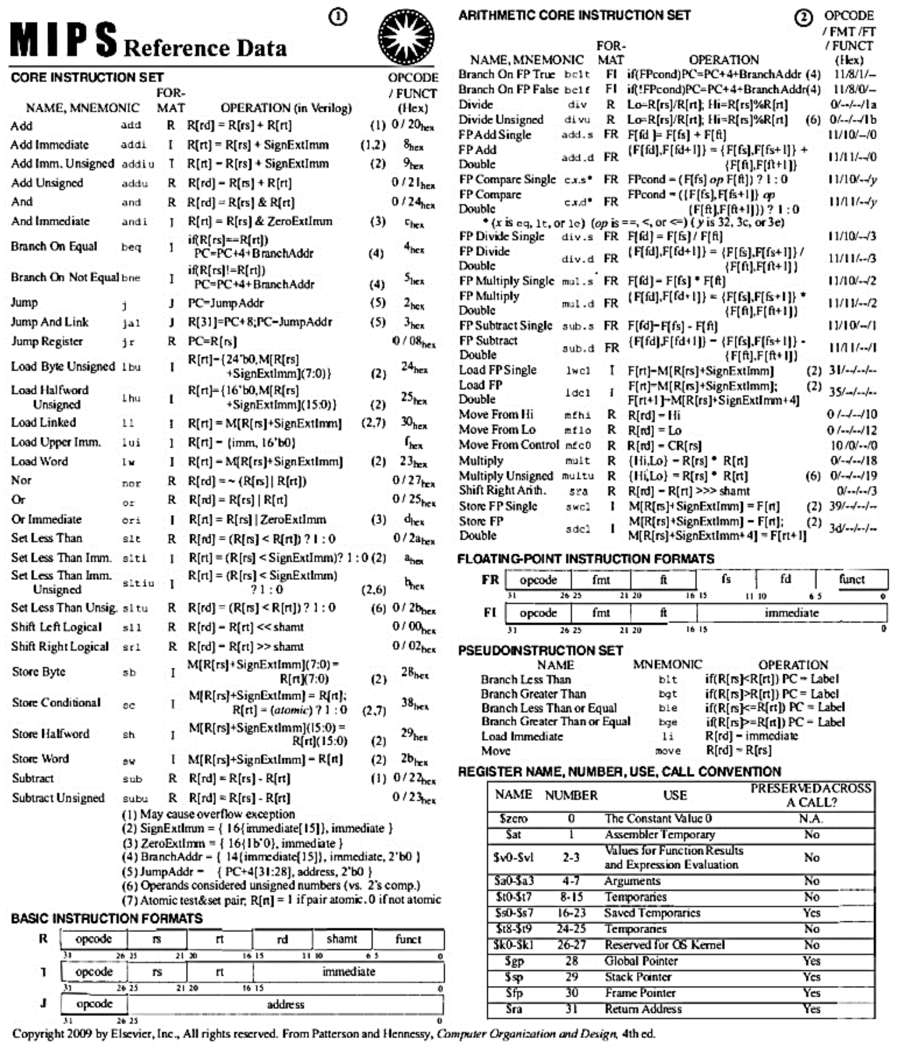
- 20 MCQs (20 marks)
- 4 FITBs (20 marks)

Remember to bring:

- Student Card (or Photo-ID)
- Pencil (2B or above), eraser, writing papers, calculator

Provided:

- MIPS Reference Data Sheet (page 1)



MIPS Reference Data ①

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	$0 / 20_{hex}$
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		$0 / 21_{hex}$
And	and	R	R[rd] = R[rs] & R[rt]		$0 / 24_{hex}$
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		$0 / 08_{hex}$
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		$0 / 27_{hex}$
Or	or	R	R[rd] = R[rs] | R[rt]		$0 / 25_{hex}$
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		$0 / 2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	$0 / 2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		$0 / 00_{hex}$
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		$0 / 02_{hex}$
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	$0 / 22_{hex}$
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		$0 / 23_{hex}$

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

	31–26	25–21	20–16	15–11	10–6	5–0
R	opcode	rs	rt	rd	shamt	funct

	31–26	25–21	20–16	15–0
I	opcode	rs	rt	immediate

	31–26	25–0
J	opcode	address

ARITHMETIC CORE INSTRUCTION SET ②

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.*x*.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.*x*.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

	31–26	25–21	20–16	15–11	10–6	5–0
FR	opcode	fmt	ft	fs	fd	funct

	31–26	25–21	20–16	15–0
FI	opcode	fmt	ft	immediate

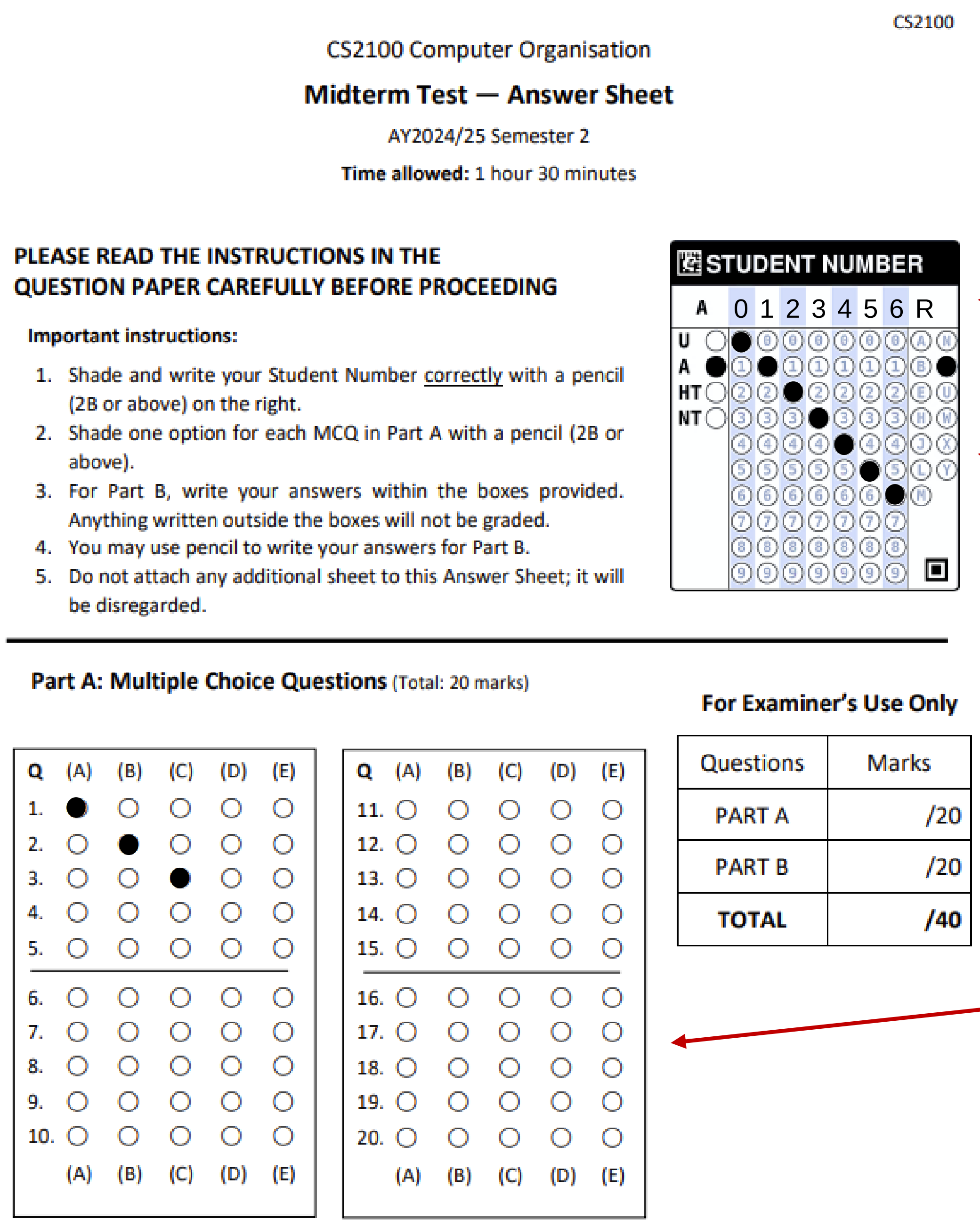
PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

Answer Sheet (page 1)



CS2100

CS2100 Computer Organisation

Midterm Test — Answer Sheet

AY2024/25 Semester 2

Time allowed: 1 hour 30 minutes

PLEASE READ THE INSTRUCTIONS IN THE QUESTION PAPER CAREFULLY BEFORE PROCEEDING

Important instructions:

1. Shade and write your Student Number <u>correctly</u> with a pencil (2B or above) on the right.
2. Shade one option for each MCQ in Part A with a pencil (2B or above).
3. For Part B, write your answers within the boxes provided. Anything written outside the boxes will not be graded.
4. You may use pencil to write your answers for Part B.
5. Do not attach any additional sheet to this Answer Sheet; it will be disregarded.

STUDENT NUMBER

	A	0	1	2	3	4	5	6	R	
U	○	●	0	0	0	0	0	0	A	N
A	●	1	●	1	1	1	1	1	B	●
HT	○	2	2	●	2	2	2	2	E	U
NT	○	3	3	3	●	3	3	3	H	W
		4	4	4	4	●	4	4	J	X
		5	5	5	5	5	●	5	L	Y
		6	6	6	6	6	6	●	M	
		7	7	7	7	7	7	7		
		8	8	8	8	8	8	8		
		9	9	9	9	9	9	9		

Part A: Multiple Choice Questions (Total: 20 marks)

Q	(A)	(B)	(C)	(D)	(E)
1.	●	○	○	○	○
2.	○	●	○	○	○
3.	○	○	●	○	○
4.	○	○	○	○	○
5.	○	○	○	○	○
6.	○	○	○	○	○
7.	○	○	○	○	○
8.	○	○	○	○	○
9.	○	○	○	○	○
10.	○	○	○	○	○
	(A)	(B)	(C)	(D)	(E)

Q	(A)	(B)	(C)	(D)	(E)
11.	○	○	○	○	○
12.	○	○	○	○	○
13.	○	○	○	○	○
14.	○	○	○	○	○
15.	○	○	○	○	○
16.	○	○	○	○	○
17.	○	○	○	○	○
18.	○	○	○	○	○
19.	○	○	○	○	○
20.	○	○	○	○	○
	(A)	(B)	(C)	(D)	(E)

For Examiner's Use Only

Questions	Marks
PART A	/20
PART B	/20
TOTAL	**/40**

Write your Student Number here.

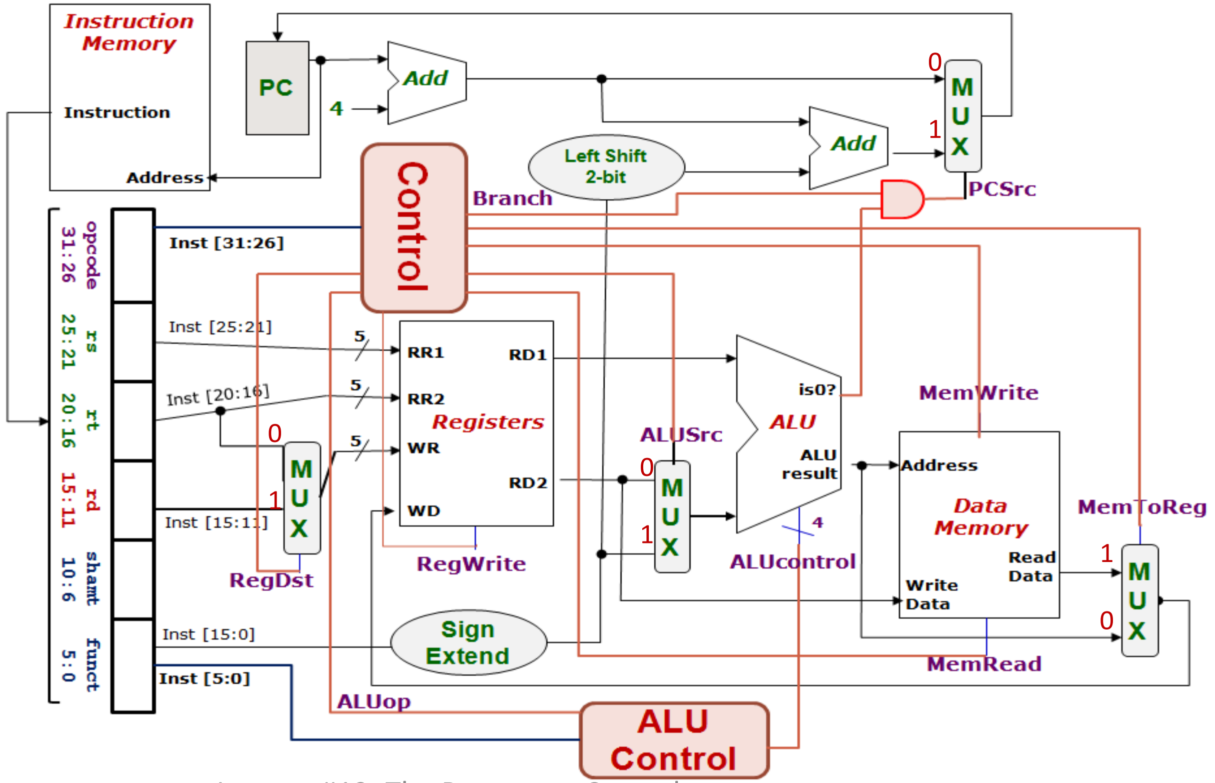
Shade your Student Number here. (<u>Check that it is correct</u>!)

Use PENCIL (2B or above)!!!

Shade your MCQ answers.

5. Control Design: **Outputs**

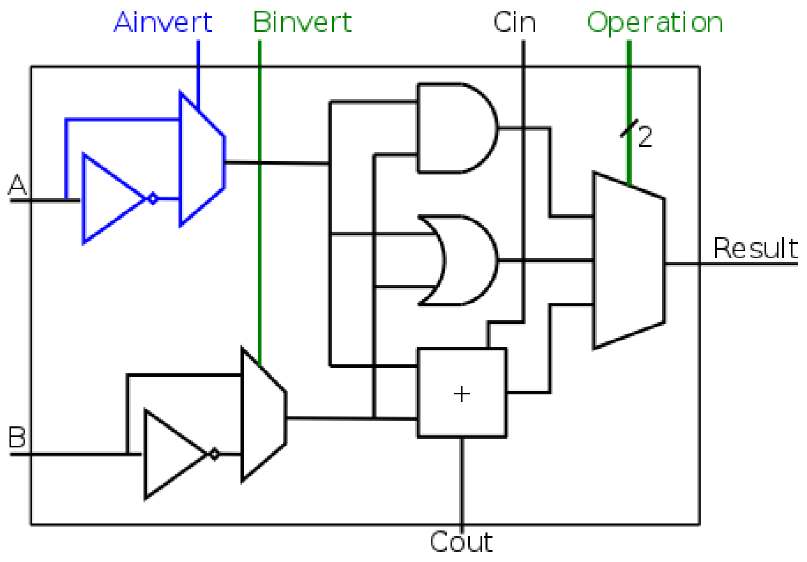
	RegDst	ALUSrc	MemTo Reg	Reg Write	Mem Read	Mem Write	Branch	ALUop	
								op1	op0
R-type	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1



5. One Bit At A Time (Aha!)

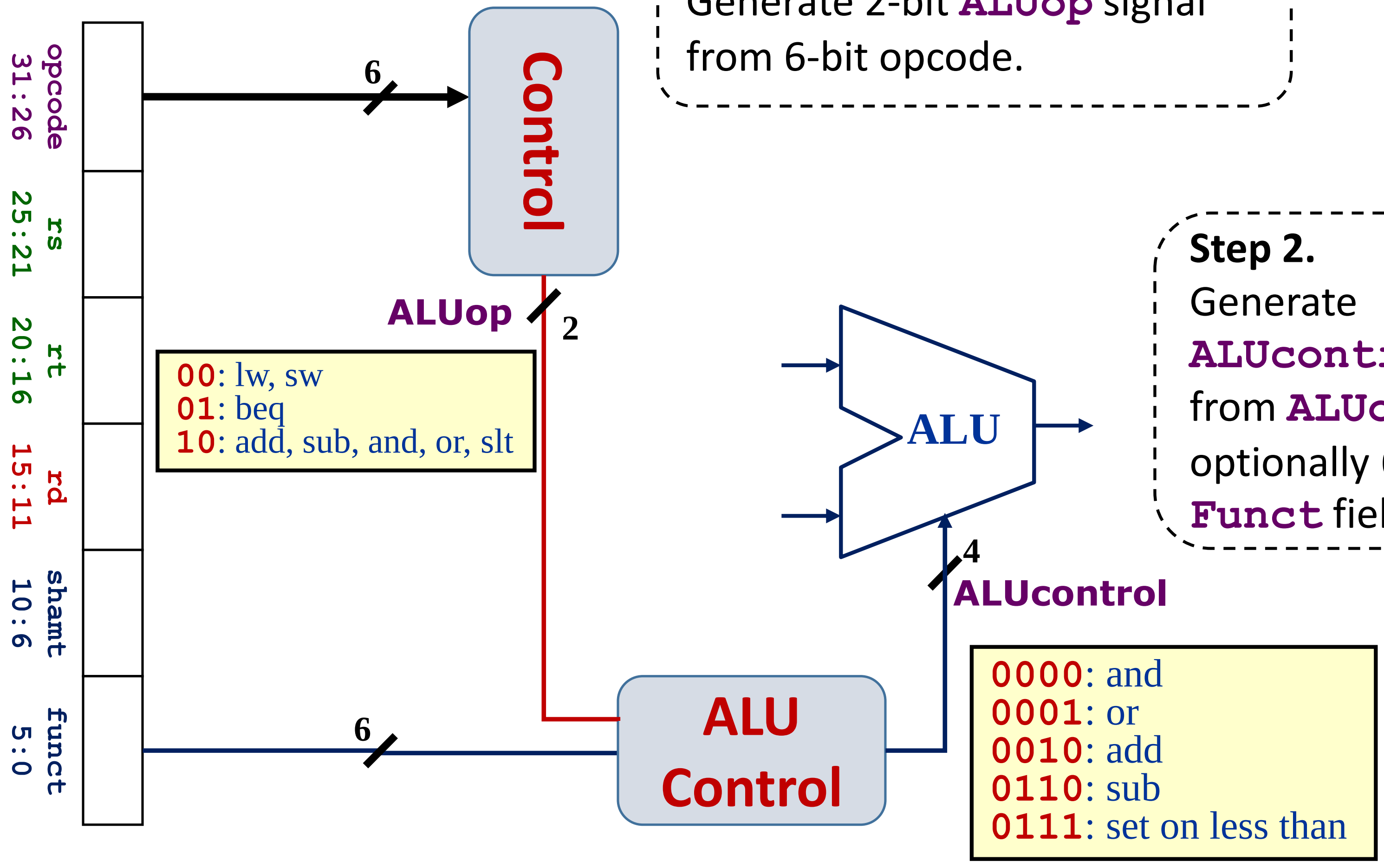
- Can you see how the **ALUcontrol** (4-bit) signal controls the ALU?
  - Note: implementation for **slt** not shown

ALUcontrol			Function
Ainvert	Binvert	Operation	
0	0	00	AND
0	0	01	OR
0	0	10	add
0	1	10	subtract
0	1	11	slt
1	1	00	NOR



Acknowledgement: Image taken from NYU Course CSCI-UA.0436

5. Two-level Implementation



00: lw, sw
01: beq
10: add, sub, and, or, slt

0000: and
0001: or
0010: add
0110: sub
0111: set on less than

Step 1.
Generate 2-bit **ALUop** signal from 6-bit opcode.

Step 2.
Generate **ALUcontrol** signal from **ALUop** and optionally 6-bit **Funct** field.

5. Generating ALUcontrol Signal

Opcode	ALUop	Instruction Operation	Funct field	ALU action	ALU control
lw	00	load word	xxxxxx	add	0010
sw	00	store word	xxxxxx	add	0010
beq	01	branch equal	xxxxxx	subtract	0110
R-type	10	add	10 0000	add	0010
R-type	10	subtract	10 0010	subtract	0110
R-type	10	AND	10 0100	AND	0000
R-type	10	OR	10 0101	OR	0001
R-type	10	set on less than	10 1010	set on less than	0111

Instruction Type	ALUop
lw / sw	00
beq	01
R-type	10

ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR

Generation of 2-bit **ALUop** signal will be discussed later

5. Design of ALU Control Unit (1/2)

- Input: 6-bit **Funct** field and 2-bit **ALUop**
- Output: 4-bit **ALUcontrol**
- Find the simplified expressions

ALUcontrol3 = 0

ALUcontrol2 = ?

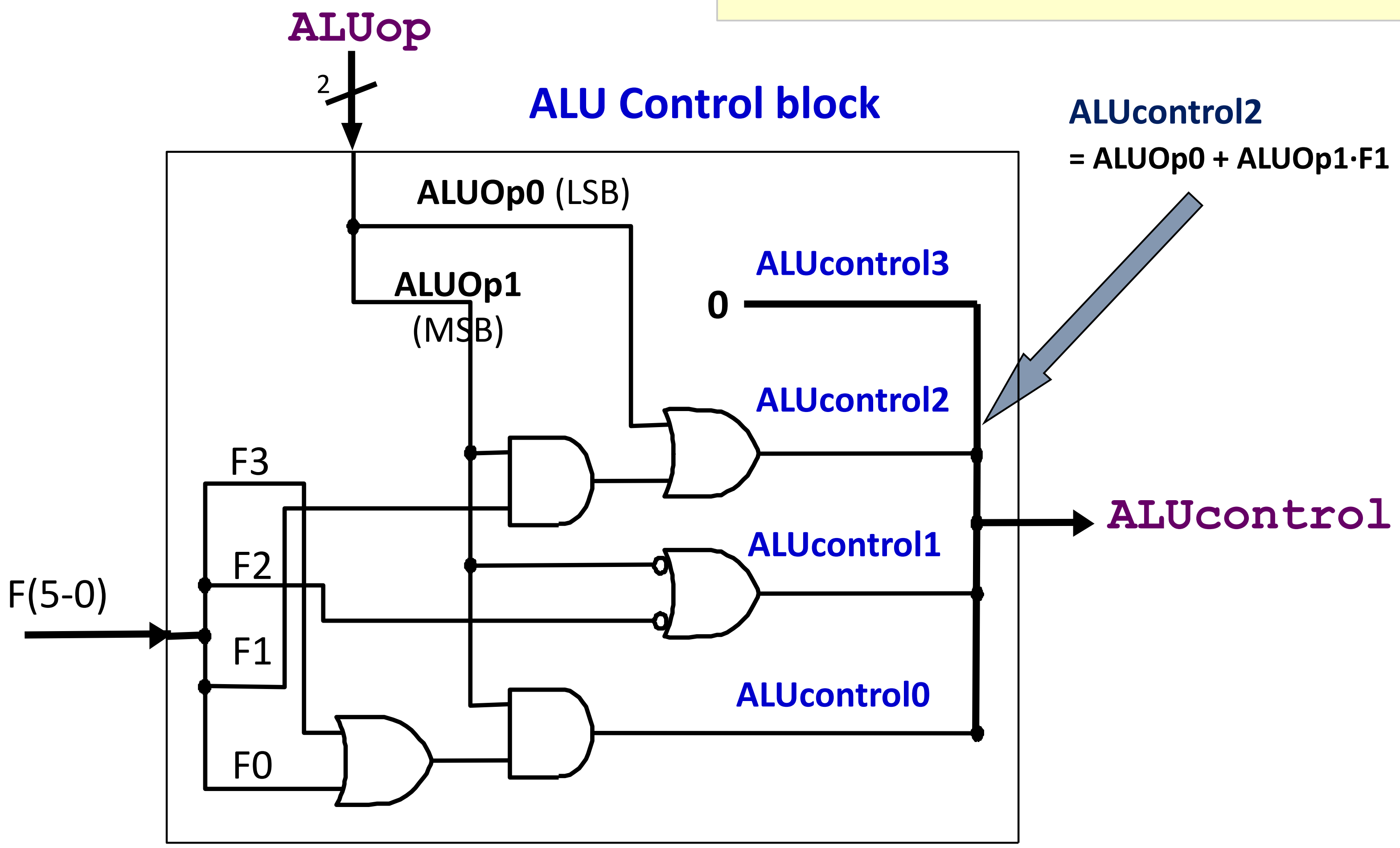
ALUop0 + ALUop1· F1

	ALUop		Funct Field (F[5:0] == Inst[5:0])						ALU control
	MSB	LSB	F5	F4	F3	F2	F1	F0	
lw	0	0	X	X	X	X	X	X	0 0 1 0
sw	0	0	X	X	X	X	X	X	0 0 1 0
beq	~~0~~ X	1	X	X	X	X	X	X	0 1 1 0
add	1	~~0~~ X	~~1~~ X	~~0~~ X	0	0	0	0	0 0 1 0
sub	1	~~0~~ X	~~1~~ X	~~0~~ X	0	0	1	0	0 1 1 0
and	1	~~0~~ X	~~1~~ X	~~0~~ X	0	1	0	0	0 0 0 0
or	1	~~0~~ X	~~1~~ X	~~0~~ X	0	1	0	1	0 0 0 1
slt	1	~~0~~ X	~~1~~ X	~~0~~ X	1	0	1	0	0 1 1 1

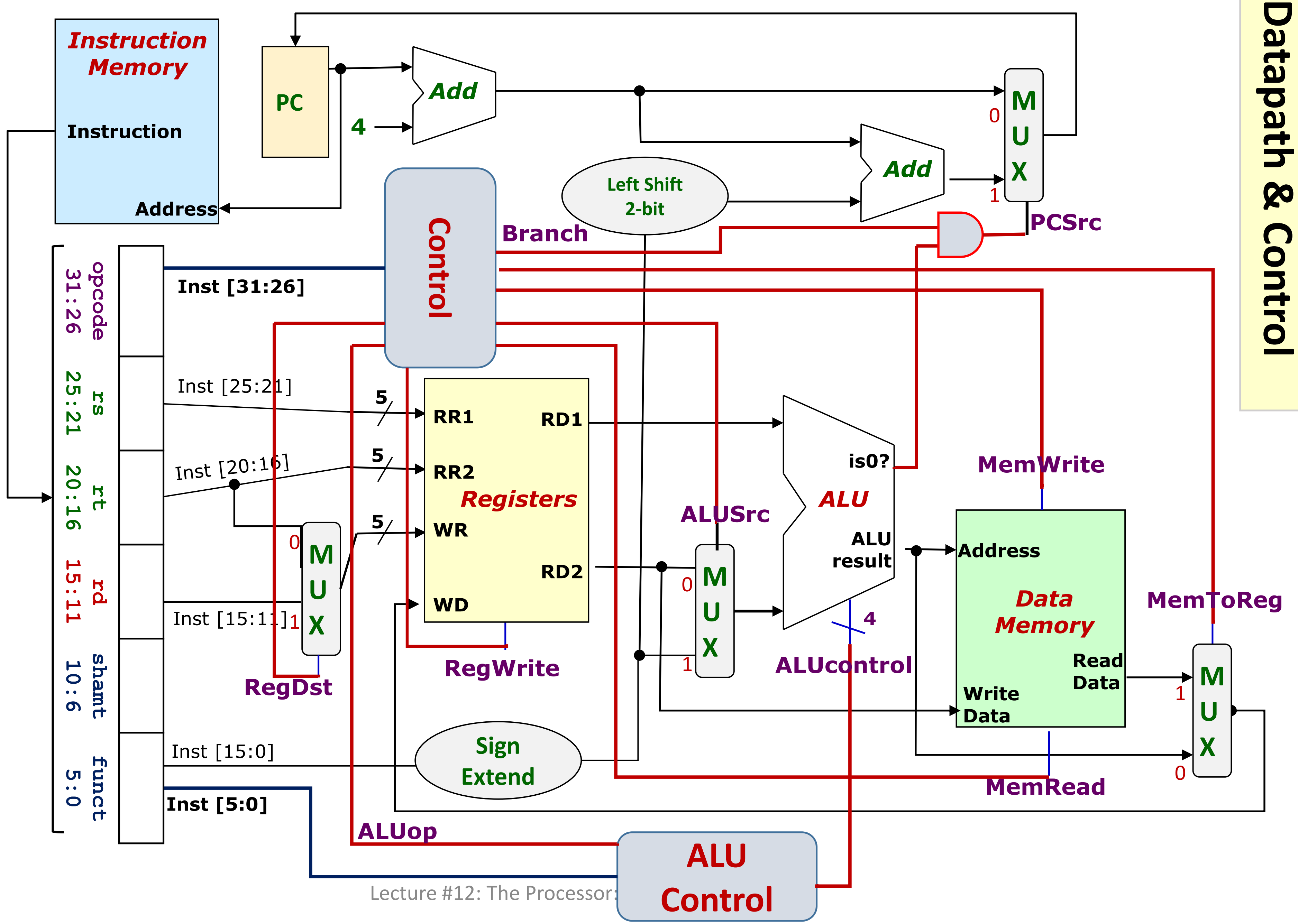
5. Design of ALU Control Unit (2/2)

- Simple combinational logic

Note: We will revisit this when we cover combinational circuits later.



Datapath & Control



AY2024/25 Semester 1

AY2024/25 Semester 1

Question 1

What is the result of the following subtraction in 8-bit signed magnitude representation:

$$00110010_{sm} - 10111101_{sm}$$

A. 10010000
B. 10010001
[✓ C. 01101111]
D. 11101110
E. None of the above

$$\begin{array}{rl} & 00110010_2 \\ + & 00111101_2 \\ = & 01101111_2 \end{array}$$

Question 2

Bob decided to design a new processor that works with 137 bits integers. He decided also to use two's complement to represent signed integer. Which of the following in hexadecimal is the largest *positive* integer that can exist in his representation system?

[✓ A. 0x0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF]
B. 0x1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
C. 0x3FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
D. 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
E. None of the above

Taking away one bit for the sign bit, we are left with 136 bits. This gives nicely 34 hexadecimals 'F's – which will be the largest positive integer.

AY2024/25 Semester 1

Question 3

This is related to Assignment 1 Question 2. Consider the following:

$$3a79_X - 2035_Y = 1792_Z$$

Which one of the following is **not** a valid solution?

A. $X = 12$, $Y = 13$ and $Z = 11$
B. $X = 15$, $Y = 16$ and $Z = 14$
C. $X = 11$, $Y = 12$ and $Z = 10$
[✓ D. $X = 13$, $Y = 15$ and $Z = 11$]
E. None of the above

No easy way. Have to try every option.

(A) $3a79_{12} = 6717_{10}$; $2035_{13} = 4438_{10}$; $6717 - 4438 = 2279$; $1792_{11} = 2279_{10}$

(B) $3a79_{15} = 12489_{10}$; $2035_{16} = 8245_{10}$; $12489 - 8245 = 4244$; $1792_{14} = 4244_{10}$

(C) $3a79_{11} = 5289_{10}$; $2035_{12} = 3297_{10}$; $5289 - 3297 = 1792$; $1792_{10} = 1792_{10}$

(D) $3a79_{13} = 8381_{10}$; $2035_{15} = 6800_{10}$; $8381 - 6800 = 1581$; $1792_{11} = 2279_{10}$

Question 4

Which of the following in hexadecimal would represent the smallest positive normalized floating point number representable in the IEEE Standard 754 single precision (32-bit) floating point?

A. 0x00000001
[✓ B. 0x00800000]
C. 0x00800001
D. 0x01000000
E. None of the above

Sign = 0; Exponent = 1; Mantissa = 0.

0b0 00000001 00000000000000000000000

= 0x00800000

(Note: All 0s are reserved for special value 0.)

During the Zoom session, a student brought up that exponent = 0 represents special value. He is right. Exponent = 0 or 255 represent special values like 0, infinity, NAN, etc.
Therefore the answer is not (A).

AY2024/25 Semester 1

Question 5

What value does the hexadecimal 0xBEA00000 represent in the IEEE Standard 754 single precision (32-bit) floating point?

A. 0.12565
B. -0.12565
C. 0.5125
[✓ D. -0.3125]
E. None of the above

0xBEA00000 = 0b1011 1110 1010 0000 0000 0000 0000 0000
Exponent = 125 – 127 = -2.
$-1.01_2 \times 2^{-2} = -0.0101_2 = -0.3125$

Question 6

Which of the following is the smallest positive base-10 value that is representable as a 32-bit integer but cannot be represented precisely as in IEEE Standard 754 single precision (32-bit) floating point? In other words, when one converts the said value (let's call it A) into the IEEE Standard 754 single precision (32-bit) floating point representation (let's call it F), and then convert F back into an integer (let's call it B), then A will not be equal to B.

A. 16777215_{10}
B. 16777216_{10}
[✓ C. 16777217_{10}]
D. 16777218_{10}
E. None of the above

We have only 23 bits in the mantissa of the IEEE standard 754 to represent an integer. We do have an additional implicit "1". So if we have a number (in binary) that starts with a '1' followed by 23 '0's – and then a '1' (a 25-bit number), then even utilizing the implicit '1' at the MSB, we still need 24 bits – one more than what we have. Hence this integer will not be representable. This number is 0x1000001, which is 16777217_{10} .

AY2024/25 Semester 1

Question 7

Which of the following is a valid C initialization statement (i.e., no compilation warning or error, and code will run accordingly)?

A. `int A[4] = {1, 2, 3, 4, 5};`
[✓ B. `int A[4] = {1, 2, 3};`]
C. `int A[4] = [1, 2, 3, 4];`
D. `int A[4] = {1, , 3, 4};`
E. None of the above

Question 8

Consider the following C program:

```
    for (int i=0; i<10; i++) {
         printf("%d ", ++i);
    }
```

What output would be printed out at the terminal after compiling and executing this C loop?

A. `1 2 3 4 5`
B. `0 2 4 6 8`
[✓ C. `1 3 5 7 9`]
D. `1 2 3 4 5 6 7 8 9`
E. None of the above

There are two increments of i.
What is printed is the pre-incremented i.

AY2024/25 Semester 1

Question 9

Consider the following C program:

```
#include <stdio.h>

int x = 1;

int main(int argc, char *argv[])
{
    int x = 2;

    for (int x=0; x<4; x++) {
        int x = 3;
        printf("%d ", x++);
    }

    printf("%d\n", x);
}
```

The innermost printf() prints its local variant of x – which is always initialized to 3. So 4 “3”s will be printed, even though x is post-incremented in the printf() because each time round this loop, a new x is instantiated. The final printf() prints the variant of x that is local to main() and which scopes out the global x. Hence it prints “2”.

What output would be printed out at the terminal after compiling and executing this program?

[✓ A. 3 3 3 3 2]
B. 3 3 3 3 3
C. 3 4 5 6 7
D. 0 3 4 6 8
E. None of the above

AY2024/25 Semester 1

Question 10

Suppose a small modification is made to the C program from Question 9 (underlined below):

```
#include <stdio.h>

int x = 1;
int main(int argc, char *argv[])
{
    int x = 2;

    for (int x=0; x<4; x++) {
        static int x = 3;
        printf("%d ", x++);
    }
    printf("%d\n", x);
}
```

Here, the big difference is in the innermost printf() prints its local variant of x is now declared as “static”. This means that there shall be only one instance of it. This will result in this same instance being incremented each time round the loop.

What output would be printed out at the terminal after compiling and executing this program?

A. 3 3 3 3 4
B. 3 4 5 6 7
C. 0 1 2 3 4
[✓ D. 3 4 5 6 2]
E. None of the above

AY2024/25 Semester 1

Question 11

Consider the following C program:

```
#include <stdio.h>

int A[4] = {1, 2, 3, 4};

int main(int argc, char *argv[])
{
    int x, y;
    int *p[2], **q;

    x = A[0];
    y = A[1];

    p[0] = &(A[1]);
    p[1] = &(A[3]);
    q = p;

    *q++ = &(A[2]);
    **q = 100;

    printf("%d %d %d %d\n", A[0], A[1], A[2], A[3]);
}
```

1. **q** was initially pointing at **p[0].**
2. After the "***q++**", it points to **p[1]** (while modifying p[0] to point to **A[2]**) which contains the address of **A[3]**.
3. The double dereferencing therefore changed what **p[1]** points to which is namely, **A[3]**.

What output would be printed out at the terminal after compiling and executing this program?

A. 1 2 3 4
[✓ B. 1 2 3 100]
C. 1 2 100 4
D. 1 100 3 4
E. None of the above

AY2024/25 Semester 1

Question 12

What is the hexadecimal encoding for the following instruction:

```
addi $t1, $sp, -120
```

A. `0x23A90088`
[✓ B. `0x23A9FF88`]
C. `0x239AFF88`
D. `0x293A0088`
E. None of the above

addi = 0b001000; \$sp = \$29 = 0b11101; \$t1 = \$9 = 0b01001;
-120 = 0b1111111110001000 = 0xFF88

Question 13

Give the hexadecimal encoding for the following branch instruction:

```
bne $a0, $t8, EXIT
```

Assume that this instruction is at PC = `0x401C` and `EXIT` is at PC = `0x35F0`.

A. `0x140035F0`
B. `0x1400401C`
[✓ C. `0x1498FD74`]
D. `0x1498FD75`
E. None of the above

bne = 0b000101; \$a0 = \$4 = 0b00100; \$t8 = \$24 = 0b11000
PC+4 = 0x4020. So the constant for the bne is (0x35F0 – 0x4020)/4 = 0xFD74 (16 bits two's complement).

Question 14

As a standard practice, we use an `ori` instruction to set a register with the 16-bit immediate zero extended to 32 bits, after a `lui` instruction has loaded the upper 16 bits. Which of the following MIPS instructions (and they are all legitimate ones) can also be used instead of the `ori` instruction to do the same thing?

A. `addi`
B. `andi`
C. `slti`
D. `nor`
[✓ E. None of the above]

We cannot use **addi** because the immediate will be sign extended, which may result in '1's in the upper 16 bits and this will affect the upper bits.
We cannot use **andi** because the lower 16 bits after **lui** is 0. And **and**'ing anything to it will still be zero.
We cannot use **slti** because the result is a comparison true (1) or false (0).
We cannot use **nor** because there is no immediate. We will need additional operations and that will complicate things.

AY2024/25 Semester 1

Question 15

Consider the following MIPS instruction given in hexadecimals:

`0x08012348`

Assuming that the PC of this instruction is **`0x2FFFFFFC`**, what would be the PC after the execution of this instruction?

A. `0x20012348`
B. `0x2F048D20`
[✓ C. `0x30048D20`]
D. `0x3001234C`
E. None of the above

This is a j instruction.
The address bits from the instruction are 0x012348. We multiply this by 4 to get 0x48D20.
PC+4 = 0x30000000. Appending this to the upper 4 bits of the PC yields the resultant PC of 0x30048D20.

Question 16

According to the MIPS reference data sheet, the last occupied location on the top of the stack is pointed to by **`$sp`** and is word aligned. Which of the following would implement a pseudo stack push instruction that pushes the content of $x onto the stack?

A. `sw $x, -4($sp)`

[✓ B. `addi $sp, $sp, -4`
`sw $x, 0($sp)`]

C. `sw $x, -4($sp)`
`addi $sp, $sp, 4`

D. `addi $sp, $sp, -4`
`lw $x, 0($sp)`

E. None of the above

The stack pointer first has to be decremented (since the stack grows towards the lower addresses) to yield an empty word before a **sw** is used to store it.

AY2024/25 Semester 1

For **Questions 17 and 18**, we will assume the parameters used in the standard MIPS encoding as shown in the MIPS Reference Sheet. In particular, we will assume that there are three instruction types: R-type, I-type, and J-type instructions. We will assume that for R-type, there are two subtypes – R-type integer and R-type floating point instructions that have the opcode of `0x00` and `0x11`, respectively.

Question 17

Given the above setup, how many distinct R-type integer instructions can be encoded?

A. 32
[✓ B. 64]
C. 128
D. 256
E. None of the above

For each R-type instruction opcode, we have 6-bits of **func** code that yields 64 instructions.

Question 18

Suppose now we give up on having the shift instructions being of R-type and instead make them I-type, thus freeing up the bits used to encode the shift amounts. Further, suppose we use these free slots for integer instructions while keeping the R-type floating point operations unchanged. Now how many distinct R-type integer instructions can we encode?

A. 256
B. 1024
[✓ C. 2048]
D. 4096
E. None of the above

This would simply extend the **func** code to 11 bits and that will yield 2048 instructions

AY2024/25 Semester 1

Question 19

Consider a 16-bit fixed length instruction set. Suppose there are three types of instructions:

- Type A: 3 operands, 3 bits each.
- Type B: 2 operands, 3 bits each.
- Type C: 1 3-bit long operand
- There must be at least one distinct instruction c

What is the maximum number of instructions that can

A. 3062
B. 3584
C. 8076
D. 8114
E. None of the above

Length of opcode: Type A = 7 bits, type B = 10 bits, type C = 13 bits.

To maximize, we should have just 1 Type A, 1 Type B, and give all the opcode bits to Type C. We can have 3-tier opcode field with 7 bits for the first opcode, 3 bits for the next and another 3 bit for the third level. We can reserve 0000000_2 to be the opcode for the only Type A instruction. Then we can use 0000001000_2 for the only Type B instruction. Then the second level opcode of 0000001001_2 onwards can be used for Type C. From 0000001001_2 to 1111111111_2 there are 1014 slots. Each of these will allow for another 8 slots (000_2-111_2) at the third level. This yields 1014 x 8 = 8112. So the maximum would be 8112 + 1 Type A + 1 Type B = 8114.

Corrected answer: Type A =1; Type B = 1; Type C = 1015 $\times$ 8 = 8120; total = **8122**.

Question 20

What is the minimum number of instructions that can be formed using the exact specification as in Question 19, and all opcodes are used?

A. 126
B. 132
C. 231
D. 4867
E. None of the above

Type A: 0000000_2 to 1111110_2 → 127 instructions.

Type B: 1111111 000 to 1111111 110 → 7 instructions.

Type C: 1111111 111 xxx → 8 instructions.

Total: 127 + 7 + 8 = **142** instructions.

AY2024/25 Semester 1

Question 21

In the MIPS datapath taught in class, which of the following statements is false?

A. The datapath to RR2 consists of 5-bits as you have 32 registers in total.
[✓ B. RD2 is used by all instructions.]
C. The ALU is a combinational circuit.
D. In the 1-bit ALU, C_{in} is only used for the + operation.
E. **`$zero`** is a register that always returns the value zero and has no effect when it is written to.

RD2 is not used by the lw and sw instructions.

Question 22

In the MIPS datapath taught in class, let us assume that `ALUoutput` is the output from the ALU. In the following, we check the value of `MemToReg` and then accordingly write to the register file.

```
WriteData = MemToReg ? Memory[ALUoutput] : ALUoutput
```

If we change the above datapath to the following:

```
WriteData = ALUoutput
```

which one of the following instructions would no longer work?

A. `sw $s1, 2($s3)`
B. `addi $s1, $s3, -50`
[✓ C. `lw $s1, 2($s3)`]
D. `j 0x12345678`
E. None of the above

The lw instruction needs the value from memory and can't just do with the ALUoutput.

AY2024/25 Semester 1

Question 23

In the MIPS datapath taught in class, which of the following statements is true?

A. The PC is a special register that is 36-bits long.
B. **`ALUcontrol`** has 6-bits as its input and 4-bits as its output.
C. **`ALUop`** is responsible for controlling the control signal **`RegDst`**.
[✓ D. The PC is updated on the rising edge of the next clock cycle.]
E. The PC is updated on the falling edge of the next clock cycle.

From the definition of the clock and when the update to PC happens during the cycle.

Question 24

In the MIPS datapath taught in class, why are the inputs to the **`MemToReg`** multiplexer reversed?

A. This is so that there is some variety in the datapath.
B. The **`MemToReg`** multiplexer is the only one that deals with memory.
[✓ C. This is so that the wires do not cross over each other in the diagram.]
D. It is because for the `sw` instruction, the source and destination registers are swapped
E. None of the above.

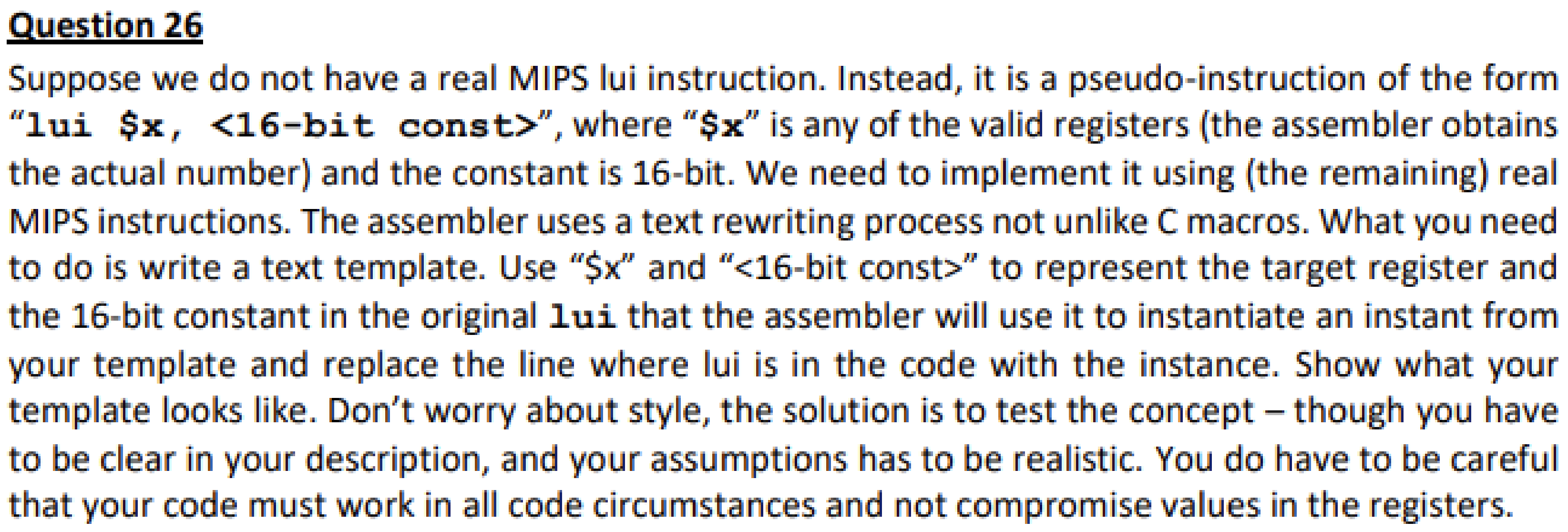
This is mainly to do with the diagram and to keep it clean.

Question 25

In the MIPS datapath taught in class, which of the following statements is false?

A. MIPS has a total of 32 registers.
B. MIPS opcodes are all 6 bits long.
C. The **`funct`** field is sometimes used to set the **`ALUcontrol`** signal.
D. The **`ALUop`** signal for the **`beq`** instruction is **`01`**.
[✓ E. None of the above.]

AY2024/25 Semester 1



Question 26

Suppose we do not have a real MIPS lui instruction. Instead, it is a pseudo-instruction of the form "**`lui $x, <16-bit const>`**", where "**`$x`**" is any of the valid registers (the assembler obtains the actual number) and the constant is 16-bit. We need to implement it using (the remaining) real MIPS instructions. The assembler uses a text rewriting process not unlike C macros. What you need to do is write a text template. Use "$x" and "<16-bit const>" to represent the target register and the 16-bit constant in the original **`lui`** that the assembler will use it to instantiate an instant from your template and replace the line where lui is in the code with the instance. Show what your template looks like. Don't worry about style, the solution is to test the concept – though you have to be clear in your description, and your assumptions has to be realistic. You do have to be careful that your code must work in all code circumstances and not compromise values in the registers.

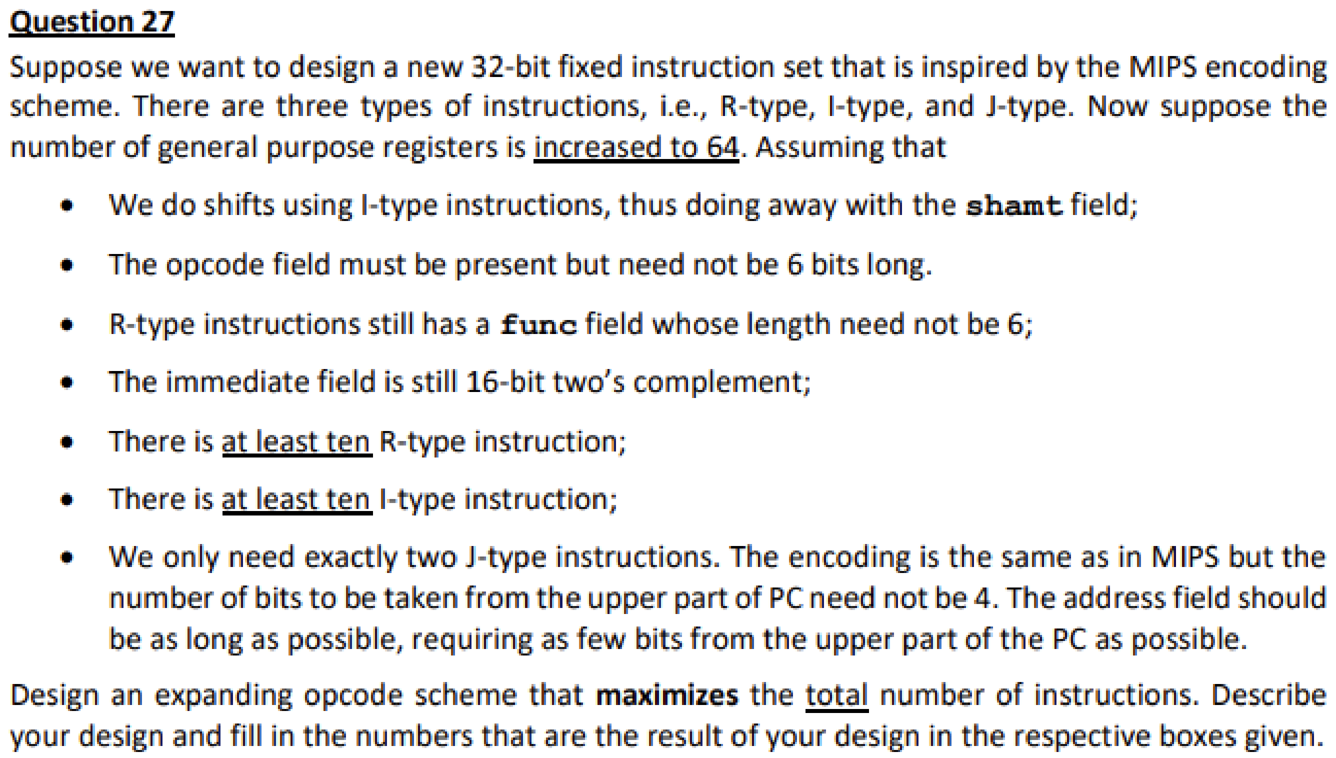
Eg: lui $t0, 0xABCD

$t0:	1010 1011 1100 1101 0000 0000 0000 0000

```
ori $x, $zero, <16-bit const>
sll $x, $x, 16
```

```
ori $t0, $zero, 0xABCD
sll $t0, $t0, 16
```

AY2024/25 Semester 1



Question 27

Suppose we want to design a new 32-bit fixed instruction set that is inspired by the MIPS encoding scheme. There are three types of instructions, i.e., R-type, I-type, and J-type. Now suppose the number of general purpose registers is <u>increased to 64</u>. Assuming that

- We do shifts using I-type instructions, thus doing away with the `shamt` field;
- The opcode field must be present but need not be 6 bits long.
- R-type instructions still has a `func` field whose length need not be 6;
- The immediate field is still 16-bit two's complement;
- There is <u>at least ten</u> R-type instruction;
- There is <u>at least ten</u> I-type instruction;
- We only need exactly two J-type instructions. The encoding is the same as in MIPS but the number of bits to be taken from the upper part of PC need not be 4. The address field should be as long as possible, requiring as few bits from the upper part of the PC as possible.

Design an expanding opcode scheme that **maximizes** the <u>total</u> number of instructions. Describe your design and fill in the numbers that are the result of your design in the respective boxes given.

Register field: 6 bits. We can then work out the constraint for the opcode field using the I-type instruction: it will need 16 bits for the immediate, 6 bits for rs and 6 bits for rt, leaving us <u>4 bits for the opcode</u>.

If **opcode** is 4 bits, and the three registers of rs, rt and rd in the R-type instruction will take up 18 bits, and since we no longer have the **shamt** field, we will therefore have 10 bits for the **func** field.

So, if we keep 10 slots of the opcode for the "at least 10" I-type instructions, 2 slots for the J-type instructions, then we will have 4 slots for R-type instructions. Since each slot in the opcode can yield 2^{10} func codes, we have 4 x 2^{10} = 4096 R-type instructions.

Number of R-type instructions:	4096
Number of I-type instructions:	10
Max. total no. of instructions:	4096 + 10 + 2 = 4108

AY2024/25 Semester 1

Question 28

Consider the MIPS datapath covered in class. For the MIPS instruction encoded as **0x10000103** fill in the corresponding elements in the boxes on the Answer Sheets. Use the notation $R to represent register number R, [$R] to represent the content of register number R and Mem(X) to represent the memory data at address X. Assume the PC value is 0x491 at the start .

RR1:	$0	Operand 1 of ALU:	[$0] or 0
WR:	$0		
WD:	0 or random value	New PC Value:	0x8A1

0x10000103 = 0b000100 00000 00000 0000 0001 0000 0011 = **beq $0, $0, 0x103**

Since $0 == $0 is true, the branch is taken,
so new PC = (PC+4) + (Immed×4) = (0x491+4) + (0x103×4) = 0x8A1

AY2023/24 Semester 2

AY2023/24 Semester 2

Q1: (1 mark) MCQ

The unsigned number 2132 in base 4 is equivalent to 185 in which base system?

A. Base-7

B. Base-8

[✓ C. Base-9]

D. Base-11

E. None of the above

$2132_4 = 158_{10} = 185_9$

Q2: (2 marks, each 0.5 mark) FITB

Convert the decimal number 85.125 into its IEEE 754 single-precision floating-point representation.

Note: For parts (a) to (c), you will enter only the binary bits and for (d), you will answer in hexadecimal format.

(a) Binary representation of the decimal number 85.125 is: 1010101.001 $= 1.010101001 \times 2^6$

(b) Exponent: 10000101

$6 + 127 = 133 = 0b10000101$

(c) Mantissa: 01010100100000000000000

(d) IEEE 754 single-precision binary representation of 85.125: 0x42AA4000

Q3: (2 marks, each 0.5 mars) FITB

0 10000100 10101110000000000000000

Convert 01000010010101110000000000000000 in the IEEE 754 single-precision floating-point representation to its decimal equivalent. Answer the following MCQs:

0b10000100 = 132; 132 – 127 = 5

a. The sign bit "0" indicates that the number is positive.

b. After converting the exponent part "10000100" to decimal and subtracting the bias, the actual exponent is 5 in decimal.

c. Including the implicit leading one, the full mantissa (fraction part) in binary format is 1.1010111

d. The decimal equivalent of the IEEE 754 representation "0 10000100 10101110000000000000000" is 53.75 in decimal.

$1.1010111_2 \times 2^5 = 110101.11_2 = 53.75$

AY2023/24 Semester 2

Q4: (1 mark) MCQ

Consider the MIPS code below:

```
main: j loop
loop: addi $t0, $t0, 1
      j main
```

Address = 0x00400004
= 0b0000 0000 0100 0000 0000 0000 0000 0100

Assuming that the first line of the program is at address 0x00400000, what is the encoded instruction for "j loop"?

A. 0x08000001
B. 0x08000004
[✓ C. 0x08100001]
D. 0x08100004
E. None of the above.

Opcode of j = 0b000010
0b000010 0000 0100 0000 0000 0000 0000 01
= 0x08100001

Q5: (1 mark) MCQ

Consider the MIPS code below:

```
main: j loop
loop: addi $t0, $t0, 1
      j main
```

Assuming **$t0** is initialized to **0**, after how many iterations of the loop will cause an overflow in register **$t0**?

A. Overflow will never occur because the MIPS architecture handles integer overflow gracefully.
[✓ B. After 2^{31} iterations, because **$t0** will overflow when it attempts to increment from $2^{31}-1$ to 2^{31}.]
C. After 2^{32} iterations, because **$t0** can hold values from 0 to 2^{32} before overflowing.
D. Overflow occurs immediately as **$t0** cannot be incremented.

Q6: (2 marks) FITB

For this question, assume size of **int** is 4 bytes, size of pointer **void*** is 4 bytes, size of **char** is 1 byte.

Program-1

```c
#include <stdio.h>

struct A {
    struct A* ptr;
    char* str;
    int num;
};

int main() {
    //create example struct A
    struct A example = {NULL, "example", 7};
    //print address of example
    printf("%p\n", &example);

    return 0;
}

```

0x7FFCC8B8

After running the above C program on a 32-bit machine, let's say the output by the printf function is 0x7FFCC8B0

What is the address of the example.num variable?

If it is not possible to tell, answer "Not possible".

AY2023/24 Semester 2

Q7: (1 mark) MCQ

For this question, assume size of **int** is 4 bytes, size of pointer **void*** is 4 bytes, size of **char** is 1 byte.

Program 2 uses the same **struct A** defined in program 1. Here, we introduce functions **changeStr** and **changeNum**.

Program-2

```c
int main() {
    // create example struct A
    struct A example = {NULL, "example", 7};
    // changeNum & changeStr
    changeNum(example, 888);
    changeStr(example, "changed");

    return 0;
}

void changeStr(struct A srctA, char* targetStr) {
    srctA.str = targetStr;
}

void changeNum(struct A srctA, int targetNum) {
    srctA.num = targetNum;
}

```

What is the final value of example after running the above C Program?

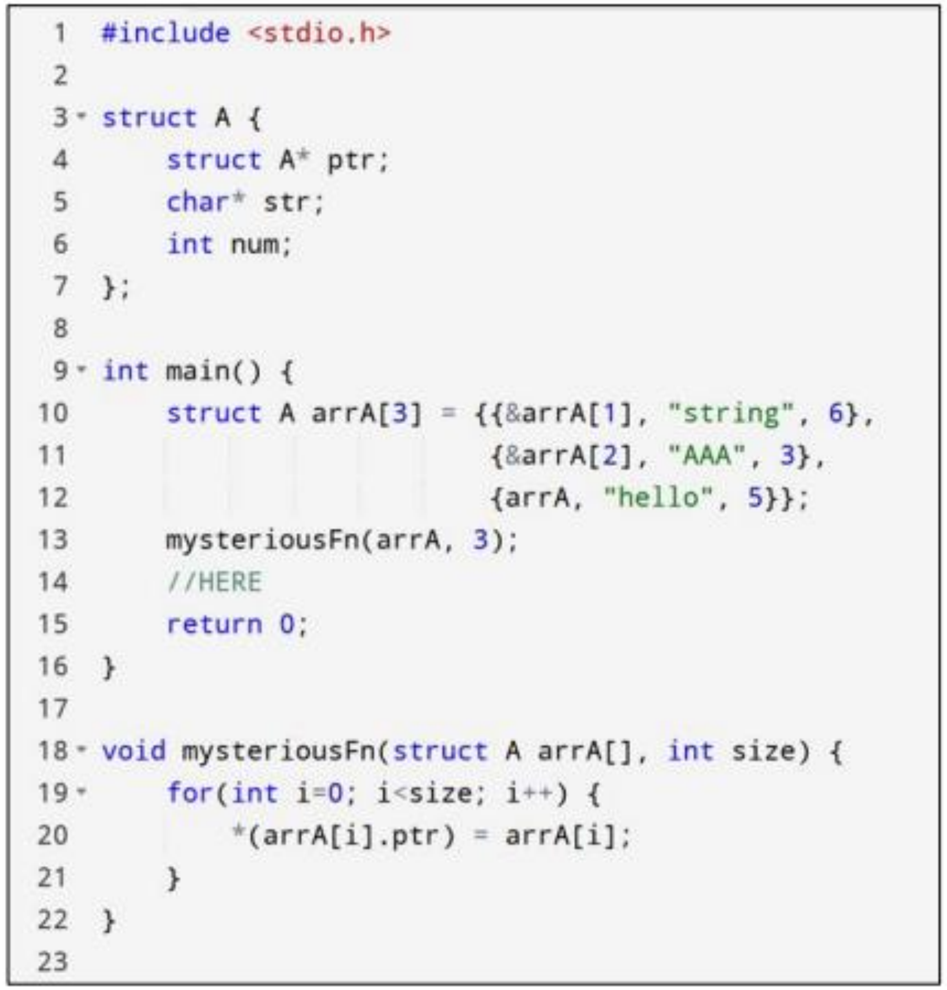
[✓ A. {NULL, “example”, 7}]
B. {NULL, “example”, 888}
C. {NULL, “changed”, 7}
D. {NULL, “changed”, 888}
E. None of the above

AY2023/24 Semester 2

Q8: (1 mark) MCQ
For this question, assume size of **`int`** is 4 bytes, size of pointer **`void*`** is 4 bytes, size of **`char`** is 1 byte.

Here, we investigate a mysterious function in the below program below:

Program 3:



```c
#include <stdio.h>

struct A {
    struct A* ptr;
    char* str;
    int num;
};

int main() {
    struct A arrA[3] = {{&arrA[1], "string", 6},
                         {&arrA[2], "AAA", 3},
                         {arrA, "hello", 5}};
    mysteriousFn(arrA, 3);
    //HERE
    return 0;
}

void mysteriousFn(struct A arrA[], int size) {
    for(int i=0; i<size; i++) {
        *(arrA[i].ptr) = arrA[i];
    }
}

```

At the point marked 'HERE' in main(), immediately following the call to **mysteriousFn** with **arrA**, what are the values of the **str** and **num** members for each struct A in **arrA**? Disregard the **ptr** member values of struct A, which are denoted by '_'. Choose the option that accurately reflects the state of **arrA**:

```
A. {{_, "string", 6}, {_, "string", 6}, {_, "hello",  5}}
B. {{_, "AAA",    3}, {_, "hello",  5}, {_, "string", 6}}
C. {{_, "hello",  5}, {_, "string", 6}, {_, "AAA",    3}}
[✓ D. {{_, "hello",  5}, {_, "string", 6}, {_, "hello",  5}}]
E. {{_, "string", 6}, {_, "string", 6}, {_, "string", 6}}
```

Q9: (1 mark) MCQ

Consider the subtraction: $(1011101)_{1s}$ - $(1001100)_{1s}$.
What is the answer in 1's complement?

A. $(0010000)_{1s}$
[✓ B. $(0010001)_{1s}$]
C. $(1101110)_{1s}$
D. $(1101111)_{1s}$
E. None of the above.

```
    1011101
+   0110011
=   0010000
+         1
=   0010001
```

Q10: (1 mark) FITB

Consider the decimal value **31.434**. What is its value in quaternary (base 4), correct to 3 quaternary digits?

$0.434_{10} = 0.1233_4 = 0.130_4$

$31.434_{10} = \mathbf{133.130_4}$

Q11: (3x1 = 3 marks) FITB

Consider the following MIPS program and assume that the program is correct and can run to completion. Answer the following questions:

```
       xor  $1, $1, $1          # instr1
       addi $1, $1, 0x0A        # instr2
       addi $2, $1, 0x01        # instr3
Loop:  andi $3, $2, 0x01        # instr4
       beq  $3, $zero, Exit     # instr5
       srl  $2, $2, 0x01        # instr6
       j    Loop                # instr7
Exit:
```

$3 always gets the LSB of the $2, $2 is 0b1011. Every iteration we right shift 1 bit of $2. Thus, after two iterations, the LSB is 0, we will stop the iteration.

(a) What is the value of $1 after executing the first instruction? **0**

(b) How many times is the code segment from instr4 to instr6 (inclusive) executed? **2**

(c) What is the value of register $2 after executing the above code? Write your answer in base 10 without any leading zero. **2**

AY2023/24 Semester 2

Q12: (1 mark) MCQ

In a typical MIPS processor's Datapath, which includes elements such as a fetch unit, decoder, register file, an Arithmetic Logic Unit (ALU), and control units. What is TRUE with respect to the function of the ALU?

A. The ALU is responsible for determining the processor's branch decisions and directing the flow of data within the datapath.

B. The ALU performs arithmetic and logical operations only on the data provided by the processor's register file.

C. The ALU serves as the main storage area for instructions that are queued for execution.

D. The ALU accelerates the connection between the processor and external devices, managing data transfers to and from memory.

[✓ E. None of the above.]

Q13: (1 mark) MCQ

Given an "ADDI" instruction in MIPS, which requires adding an immediate value to a register's content, select the sequence that correctly outlines the steps the datapath performs to execute this instruction:

[✓ A. The instruction is fetched from memory and placed into the Instruction Register (IR). The specified register's content is read and sent to one input of the ALU. The immediate value is decoded from the instruction and supplied to the other input of the ALU. The ALU executes the addition, and the resulting sum is written back into the designated register in the register file.]

B. The instruction is fetched from memory into the Register File, the immediate operand is moved into the ALU, the register is selected and its content is sent to the ALU, the ALU performs the addition, and the result is written back into the register file.

C. After fetching the instruction from memory, the immediate value is temporarily stored in the Register File. The content of the specified register is then fetched into the ALU where the addition takes place using the stored immediate value.

D. The instruction is retrieved from memory, causing the Program Counter (PC) to increment by 8, pointing to the subsequent instruction. The fetched instruction is decoded, and its operands are processed by the ALU.

(B) Not store in the Register File but in the instruction register.

(C) Immediate value is stored in the instruction not in the RF.

(D) Increment by 4, not 8.

Q14: (2x2 = 4 marks) FITB

Given MIPS instructions, find out the signals/values for the control and data along the specified paths. All provided values are in decimal. All answers should also be expressed in decimal.

(a) sw $17, 16($8) where $17 holds the value 1 and $8 holds the value 48, what are the control signals and data value at:

(i) RD2 = 1
(ii) RD1 = 48
(iii) ALUSrc = 1
(iv) MemWrite = 1

(b) 000100 01000 00000 1111111111111110 where $8 holds the value of 0 and $16 holds the value 1, what are the control signals and data values at:

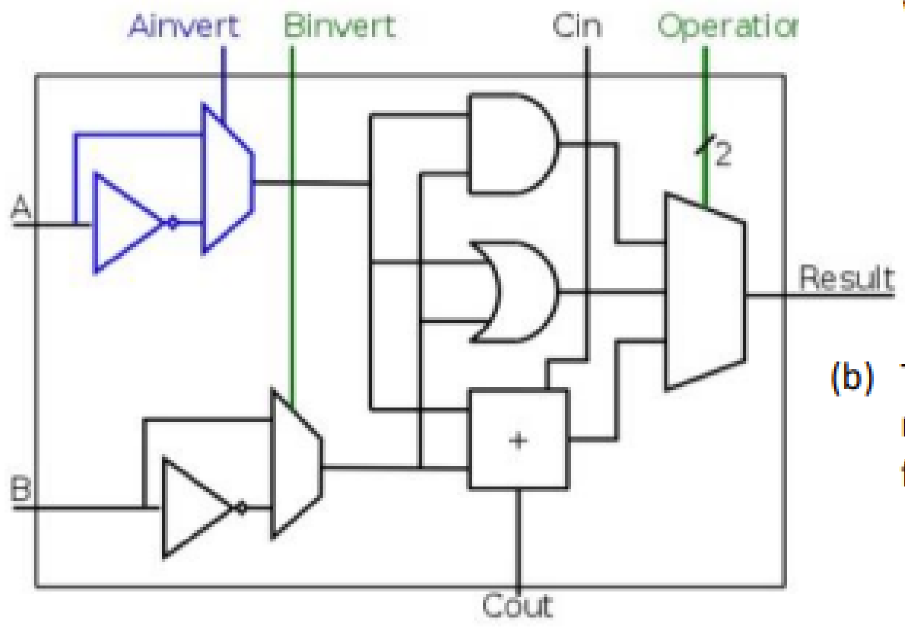
(i) RR2 = 0
(ii) ALUSrc = 0
(iii) is0? = 1
(iv) PCSrc = 1

beq $8 $0, -2

AY2023/24 Semester 2

Q15: (3x2 = 6 marks) FITB

You are implementing a 4-bit ALU based on the single-bit ALU taught during the lecture, which is also shown in the below figure.



(a) To implement the 2's complement subtraction function, what signal should you set for the Ainvert, Binvert and Cin? Consider an example where A=3 and B=2 (both in decimal), and we want to calculate A-B, what are the control signals and data values at:

(i) Ainvert = 0

(ii) Binvert = 1

(iii) Cin = 1

(iv) The Least Significant Bit of Result = 1

(b) To implement the overflow function (for the 2's complement addition/subtraction), you will need to add an additional logic gate and an additional output called Overflow. For each of the following, determine whether it is T (for True) or F (for False).

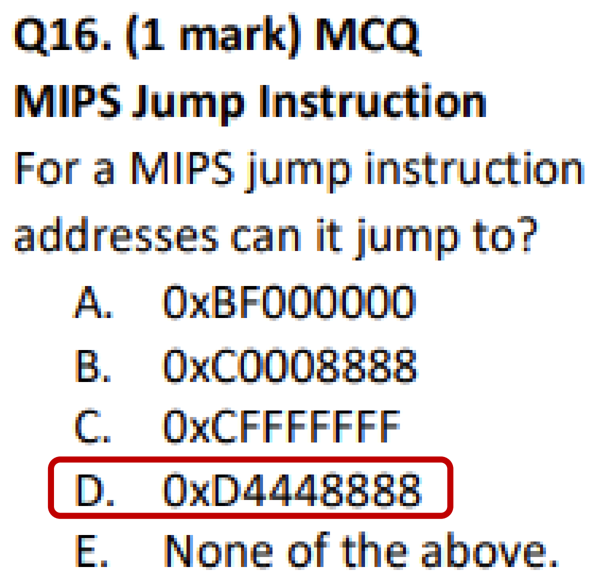
(i) Overflow = Cin XOR Cout at the Most Significant Bit. T

(ii) Overflow or not, the Result output should remain the same. T

(c) Suppose the latency of all logic gates and adder (including NOT, AND, OR, and ADDER) is 1ps (pico-second), and latency of all MUXs is 2ps, what is the maximum latency of this 4-bit CPU (excluding the implementation of overflow function in part (b))? More specifically, once the input and the control signals are ready, in the worst case, how many ps is required to obtain the complete 4-bit Result for all operations? Express your result in decimal value only, for example, if the latency is 10ps for signed addition and 11ps for signed subtraction, just answer 11.

It takes 4ps to output the LSB Cout, hence 6ps to get MSB Cin, 7 ps to get MSB ADDER output, 9ps to get MSB Result.

9



Q16. (1 mark) MCQ

MIPS Jump Instruction

For a MIPS jump instruction that is located at the address 0xCFFFFFFC, which of the following addresses can it jump to?

A. 0xBF000000

B. 0xC0008888

C. 0xCFFFFFFF

[✓ D. 0xD4448888]

E. None of the above.

If PC = 0xCFFFFFFC, then PC+4 = 0xD0000000, and hence it can only jump to addresses that start with 0b1101.

AY2023/24 Semester 2

Q17: (11 marks) FITB

ISA: Suppose that we are making an Instruction Set Architecture (ISA) with THREE classes of instructions:

Class X: contains 2 registers, and 8-bit immediate value (imm).
Class Y: contains a 12-bit jump address (addr).
Class Z: contains 2 registers and a shift amount (shamt).

The following information about the architecture are given:

- There are 6 registers which can be used.
- Each register holds a 16-bit value.
- The size of each instruction is one word, which is 16 bits for this architecture.
- The address space is 16 bits, and jumps are only done to word-aligned addresses.

The ISA contains at least the following instructions:

```
LDH $RD $RA imm
STR $RD $RA imm
JMP addr
JAL addr
SRA $RD $RA shamt
```

a. How many bits are needed to represent the maximum shift amount in this architecture? You are to follow the same design reasoning used in MIPS. (1 mark)

4 bits.

In MIPS, the shamt is 5 bits for 32-bit registers. Following the same reasoning, we have 4 bits **for the maximum shift amount as the registers here are 16-bit long.**

AY2023/24 Semester 2

Q17: (11 marks) FITB

ISA: Suppose that we are making an Instruction Set Architecture (ISA) with THREE classes of instructions:

Class X: contains 2 registers, and 8-bit immediate value (imm).
Class Y: contains a 12-bit jump address (addr).
Class Z: contains 2 registers and a shift amount (shamt).

The following information about the architecture are given:

- There are 6 registers which can be used.
- Each register holds a 16-bit value.
- The size of each instruction is one word, which is 16 bits for this architecture.
- The address space is 16 bits, and jumps are only done to word-aligned addresses.

The ISA contains at least the following instructions:

```
LDH $RD $RA imm
STR $RD $RA imm
JMP addr
JAL addr
SRA $RD $RA shamt
```

Note: From this point onwards, assume that **shamt** is a 3-bit value.

b. Suppose that the encoding space is fully utilized and we are minimizing the number of instructions in the ISA. How many instructions would be under (i) Class X, (ii) Class Y, and (iii) Class Z? (3 marks)

Class X: 3

Class Y: 3

Class Z: 8

Class X has at least 2 instructions (LDH, STR); Class Y has at least 2 instructions (JMP, JAL); Class Z has at least 1 instruction (SRA). Since there are 6 registers, we need to use 3 bits to represent a register. This leaves us with the following number of bits for the opcodes:
Class X: 2 bits for opcode, 6 bits for the two registers, 8 bits for imm
Class Y: 4 bits for opcode, 12 bits for addr
Class Z: 7 bits for opcode, 6 bits for the two registers, 3 bits for shamt

To minimize the number of instructions, we give class X the most number of opcodes possible, which is **3**.
For example, we give X the opcodes 01,10,11; leaving the opcodes starting from 00 for classes Y/Z.
We then give class Y the most number of opcodes possible. In this example, we can give **3** opcodes (eg: 0001/0010/0011) to class Y; leaving the opcodes starting from 0000 for class Z.
Class Z would have opcodes of the form 0000xxx, which adds up to **8** opcodes.

AY2023/24 Semester 2

Q17: (11 marks) FITB

ISA: Suppose that we are making an Instruction Set Architecture (ISA) with THREE classes of instructions:

Class X: contains 2 registers, and 8-bit immediate value (imm).
Class Y: contains a 12-bit jump address (addr).
Class Z: contains 2 registers and a shift amount (shamt).

The following information about the architecture are given:

- There are 6 registers which can be used.
- Each register holds a 16-bit value.
- The size of each instruction is one word, which is 16 bits for this architecture.
- The address space is 16 bits, and jumps are only done to word-aligned addresses.

The ISA contains at least the following instructions:

```
LDH $RD $RA imm
STR $RD $RA imm
JMP addr
JAL addr
SRA $RD $RA shamt
```

c. Suppose that the encoding space is fully utilized and we are maximizing the number of instructions in the ISA. How many instructions would be under (i) Class X, (ii) Class Y, and (iii) Class Z? (3 marks)

Class X: 2

Class Y: 2

Class Z: 48

To maximize the number of instructions, we restrict classes X/Y as much as possible, giving a maximum number of opcodes for classes Z.
Note that class X has at least two instructions. Reserve opcodes 00 and 01 for those instructions.
Classes Y & Z will have opcodes starting with 10/11.
Note that class Y has at least 2 instructions. Reserve opcode 1000/1001 for class Y.
This leaves the following opcodes for class Z:
1010xxx : 8 opcodes
1011xxx : 8 opcodes
11xxxxx : 32 opcodes
For a total of **48** opcodes for class Z.

AY2023/24 Semester 2

Q17: (11 marks) FITB

ISA: Suppose that we are making an Instruction Set Architecture (ISA) with THREE classes of instructions:

Class X: contains 2 registers, and 8-bit immediate value (imm).
Class Y: contains a 12-bit jump address (addr).
Class Z: contains 2 registers and a shift amount (shamt).

The following information about the architecture are given:

- There are 6 registers which can be used.
- Each register holds a 16-bit value.
- The size of each instruction is one word, which is 16 bits for this architecture.
- The address space is 16 bits, and jumps are only done to word-aligned addresses.

The ISA contains at least the following instructions:

```
LDH $RD $RA imm
STR $RD $RA imm
JMP addr
JAL addr
SRA $RD $RA shamt
```

d. Given that JMP is the jump instruction, what would be maximum jump range available for the JMP instruction? (1 mark)

A. 4096 bytes
[✓ B. 8192 bytes]
C. 16384 bytes
D. 32768 bytes
E. 65536 bytes

Since the jumps are word-aligned to 16 bits (2 bytes), we can take the address and pad one zero to the end. This results in 13-bit addresses, which translates to a jump range of 8192 bytes.

AY2023/24 Semester 2

Q17: (11 marks) FITB

ISA: Suppose that we are making an Instruction Set Architecture (ISA) with THREE classes of instructions:

Class X: contains 2 registers, and 8-bit immediate value (imm).
Class Y: contains a 12-bit jump address (addr).
Class Z: contains 2 registers and a shift amount (shamt).

The following information about the architecture are given:

- There are 6 registers which can be used.
- Each register holds a 16-bit value.
- The size of each instruction is one word, which is 16 bits for this architecture.
- The address space is 16 bits, and jumps are only done to word-aligned addresses.

The ISA contains at least the following instructions:

```
LDH $RD $RA imm
STR $RD $RA imm
JMP addr
JAL addr
SRA $RD $RA shamt
```

From part (b):
Class X: 2-bit opcode
Class Y: 4-bit opcode
Class Z: 7-bit opcode

e. By changing the number of bits of address in Class Y instructions, maximize the jump range available for the JMP instruction. Make sure that all modifications would still allow the implementation of Class X and class Z instructions in the ISA. Write your answer as a single decimal number without leading zero. (3 marks)

I. What would be the maximum jump range (in number of bytes) available after your modification?

16384

II. What is the maximum number of instructions possible in the ISA after your modification, assuming that the encoding space is fully utilized?

36

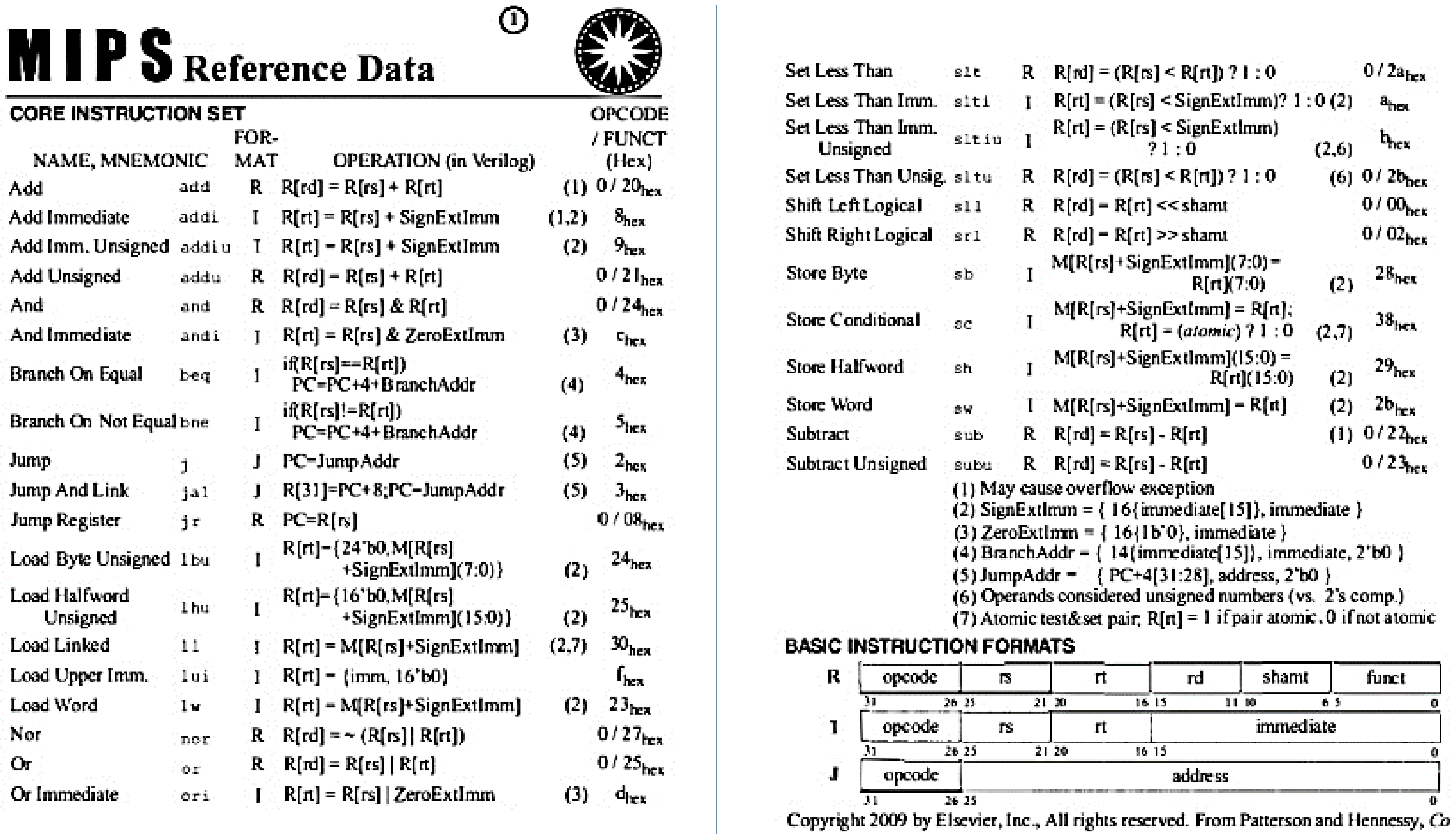
III. What is the minimum number of instructions possible in the ISA after your modification, assuming that the encoding space is fully utilized?

21

If we were to make the addr 14 bits, we would not be able to fit all the instructions into 16 bits. To show this, suppose we make addr into 14 bits. Then we only have 2 bits for the opcode. Since both classes X and Y have 2 bits of opcode, and each of them have at least 2 instructions, the encoding space would have no space for class Z instructions.
We can make addr into at most 13 bits. One possible opcode combination for this is:
Class X: 00/01
Class Y: 100/101
Class Z: 11xxxxx
This would result in a jump range of 13+1 = 14 bits, which is **2^{14} or 16384 bytes**

Maximum: 2 (class X) + 2 (class Y) + 2×2^4 (class Z) = **36 opcodes.**

Minimum: 2 (class X) + 3 (class Y) + 2^4 (class Z) = **21 opcodes.**



①

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	$0 / 20_{hex}$
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		$0 / 21_{hex}$
And	and	R	R[rd] = R[rs] & R[rt]		$0 / 24_{hex}$
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		$0 / 08_{hex}$
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		$0 / 27_{hex}$
Or	or	R	R[rd] = R[rs] | R[rt]		$0 / 25_{hex}$
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		$0 / 2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	$0 / 2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		$0 / 00_{hex}$
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		$0 / 02_{hex}$
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	$0 / 22_{hex}$
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		$0 / 23_{hex}$

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26	25 21	20 16	15 11	10 6	5 0
I	opcode	rs	rt	immediate		
	31 26	25 21	20 16	15 0		
J	opcode	address				
	31 26	25 0				

END OF FILE