Computer Organization Review

I. Final Grades

- Daily Performance (Homeworks?): **15%**
- Final Exam: **85%**

II. Final Exam Format

1. 20 Points -> Basic Knowledge; Multiple Choice & Fill in the Blanks (10 Questions)
2. 25 Points -> Combinatorial Logic & Sequential Logic
3. 15 Points -> Instruction System & Assembly
4. 25 Points -> Storage System (Main Memory + Cache + Virtual Memory)
5. 15 Points -> CPU Analysis

III. Review

A. Combinatorial Logic

i. Basics

- Logic Gates:

1.	NOT:	Opposite of everything
2.	AND:	Only 1 when 1-1
3.	OR:	Only 0 when 0-0
4.	NAND:	Opposite of AND
5.	NOR:	Opposite of OR
6.	XOR:	Only 1 when 1-0 | 0-1

- Binary Calculation Laws:

1.	Zero Law:	$A \vee 1 \Leftrightarrow 1$	$A \wedge 0 \Leftrightarrow 0$	
2.	Idempotent law:	$A \vee A \Leftrightarrow A$	$A \wedge A \Leftrightarrow A$	
3.	Law of identity:	$A \vee 0 \Leftrightarrow A$	$A \wedge 1 \Leftrightarrow A$	$A \oplus 0 \Leftrightarrow A$
4.	Law of contradiction:		$A \wedge \neg A \Leftrightarrow 0$	
5.	Law of Middle:	$A \vee \neg A \Leftrightarrow 1$		
6.	Exchange law:	$A \vee B \Leftrightarrow B \vee A$	$A \wedge B \Leftrightarrow B \wedge A$	$A \oplus B \Leftrightarrow B \oplus A$
7.	Binding law:	$(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$		
		$(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$		
		$(A \oplus B) \oplus C \Leftrightarrow A \oplus (B \oplus C)$		
8.	Distribution law:	$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$		
		$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$		
		$A \wedge (B \oplus C) \Leftrightarrow (A \wedge B) \oplus (A \wedge C)$		
9.	Law of absorption:	$A \vee (A \wedge B) \Leftrightarrow A$	$A \wedge (A \vee B) \Leftrightarrow A$	
10.	De Morgan Law:	$\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$		
		$\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$		
11.	Double negation:	$\neg\neg A \Leftrightarrow A$		

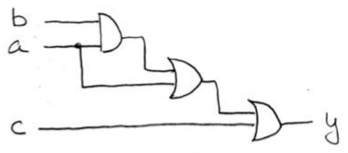
12. Hypothetical translocation: $A \longrightarrow B \Leftrightarrow \neg B \longrightarrow \neg A$
13. $A \longrightarrow B \Leftrightarrow \neg A \vee B$
14. $A \oplus A \Leftrightarrow 0$
15. $A \oplus 1 \Leftrightarrow \neg A$
16. $A \leftrightarrow B \Leftrightarrow (A \longrightarrow B) \wedge (B \longrightarrow A)$
17. $A \oplus B \Leftrightarrow (\neg A \wedge B) \vee (A \wedge \neg B)$
18. $A \oplus B \Leftrightarrow \neg(A \leftrightarrow B)$

- Laws of Boolean Algebra:

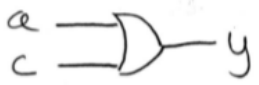
1.	$x \cdot \bar{x} = 0$	$x + \bar{x} = 1$
2.	$x \cdot 0 = 0$	$x + 1 = 1$
3.	$x \cdot 1 = x$	$x + 0 = x$
4.	$x \cdot x = x$	$x + x = x$
5.	$x \cdot y = y \cdot x$	$x + y = y + x$
6.	$(xy)z = x(yz)$	$(x + y) + z = x + (y + z)$
7.	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$
8.	$xy + x = x$	$(x + y)x = x$
9.	$\bar{x}y + x = x + y$	$(\bar{x} + y)x = xy$
10.	$\overline{x \cdot y} = \bar{x} + \bar{y}$	$\overline{x + y} = \bar{x} \cdot \bar{y}$

- Circuit Simplification

Example:



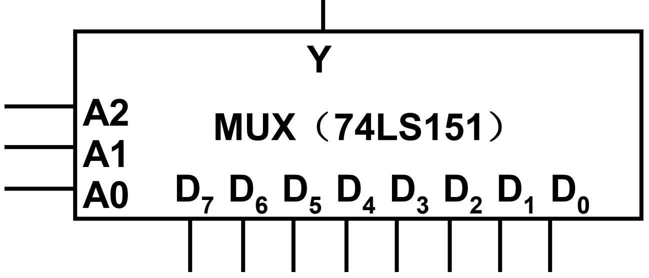
$$\begin{aligned} y &= \big((ab) + a\big) + c \\ &= ab + a + c \\ &= a(b + 1) + c \\ &= a(1) + c \\ &= a + c \end{aligned}$$



- K-Maps:

//NOT YET SUMMARIZED//

- Dunno what this is but if there's this pic



This thing is somekind of **Data Selector**, in this case choose 1 out of 8

A2~A0: Controller

D7~D0: Data Switch

Y: Output

In this case,

$$Y = \overline{A_2}\,\overline{A_1}\,\overline{A_0}D_0 + \overline{A_2}\,\overline{A_1}A_0D_1 + \overline{A_2}A_1\overline{A_0}D_2 + \overline{A_2}A_1A_0D_3 + A_2\overline{A_1}\,\overline{A_0}D_4 + A_2\overline{A_1}A_0D_5 + A_2A_1\overline{A_0}D_6 + A_2A_1A_0D_7\,(\overline{EN}=0)$$
$$= m_0D_0 + m_1D_1 + m_2D_2 + m_3D_3 + m_4D_4 + m_5D_5 + m_6D_6 + m_7D_7$$

Or can also,
A2~A0: Data Switch
D7~D0: Controller
Y: Output
In this case, example:
当 D_7 ~ D_0 为 0000_0000 时，Y=0
当 D_7 ~ D_0 为 1111_1111 时，Y=1
当 D_7 ~ D_0 为 0000_0001 时，$Y=m_0=\bar{A}\bar{B}\bar{C}$
当 D_7 ~ D_0 为 1010_0101 时，$Y=m_7+m_5+m_2+m_0$

See how Y is got by combinations of all possible situations of A2, A1, A0, Therefore can get this function:

$$F(A,B,C) = \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB$$

WHICH IS EQUALS TO

$$F = \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB(\bar{C}+C) = m_1 + m_2 + m_6 + m_7$$

THEN BY COMPARING Y and F:

D0 = 0	D1 = 1	D2 = 1	D3 = 0
D4 = 0	D5 = 0	D6 = 1	D7 = 1

ii. Exercise

1. Simplify: $\mathrm{F} = \mathrm{BC} + \bar{A}\bar{B}\bar{C} + B\bar{C}$
   Final Answer: $\mathrm{F} = \mathrm{B} + \bar{A}\bar{C}$
   Answer:
2. Simplify: $\mathrm{F} = \overline{A + \bar{A}B + \bar{A}\bar{B}} + \overline{A + \bar{B}}$
   Final Answer: $\mathrm{F} = \bar{A}B$
   Answer:
3. Simplify: $\mathrm{F} = \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C} + A\bar{B}C\bar{D} + ABD + \bar{B}C\bar{D} + \bar{A}$
   Final Answer: $\mathrm{F} = \bar{B}\bar{C} + \bar{B}\bar{D} + \bar{A} + \mathrm{BD}$
   Answer:
4. Simplify: $\mathrm{F} = \mathrm{AC} + \bar{A}\bar{B}C$

Final Answer: $\mathrm{F} = \mathrm{AC} + \bar{B}C$

Answer:

5. Simplify: $\mathrm{F} = \mathrm{ABC} + \mathrm{ABD} + \mathrm{ABE} + \mathrm{ACD} + \mathrm{ACE} + \overline{A + D + E} + \bar{B}\bar{C}D + \bar{B}\bar{C}E + \bar{B}\bar{D}\bar{E} + \bar{C}\bar{D}\bar{E}$

Final Answer: $\mathrm{A} + \bar{B}\bar{C} + \bar{D}\bar{E}$. Solve $\overline{A + D + E} = \bar{A}\bar{D}\bar{E}$ first. K-maps (Note that ABC must be expanded into 4 product terms consisting of 5 variables).

DE \ ABC	000	001	011	010	110	111	101	100
00	1	1	1	1	1	1	1	1
01	1				1	1	1	1
11	1				1	1	1	1
10	1				1	1	1	1

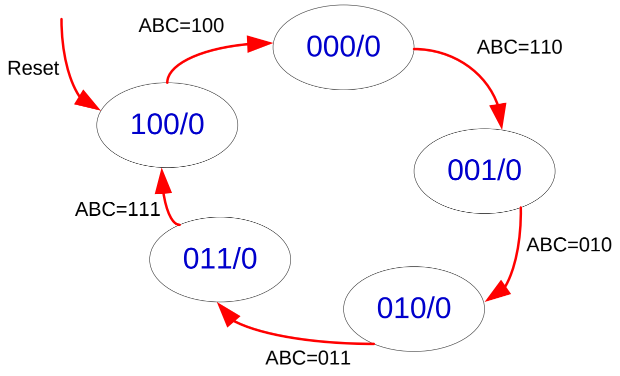
P.S. 000 columns cannot form 4 columns with 111, 101, 100!!!

Answer:

B. Sequential Logic

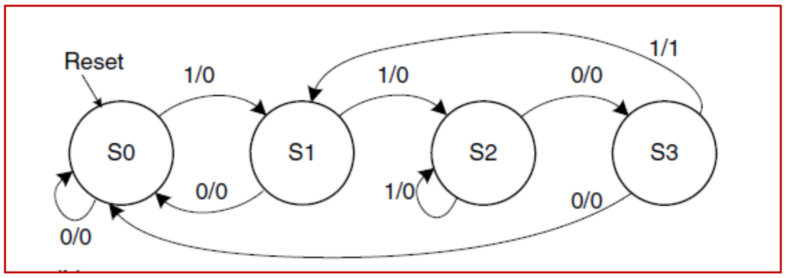
i. Basics

- Moore: There's a delay, input is saved to register first and will be printed out when it's enabled
  In Verilog, just see if the last state has been reached
  Assign zo = (state==S4) ? 1'b1 : 1'b0;



THE OUTPUTS ARE WRITTEN ON THE CIRCLES (Output based on State)

- Mealy: There's no delay, the input is directly the output at the same time
  In Verilog, the output will depend on the state before and the current input
  Assign zo = (state==S3) & (data==1'b0);



THE OUTPUT IS IMMEDIATELY WRITTEN ON THE ARROWS (Output = Input)

- To create the logic from truth table, just directly find all data that makes it True.
  Example:

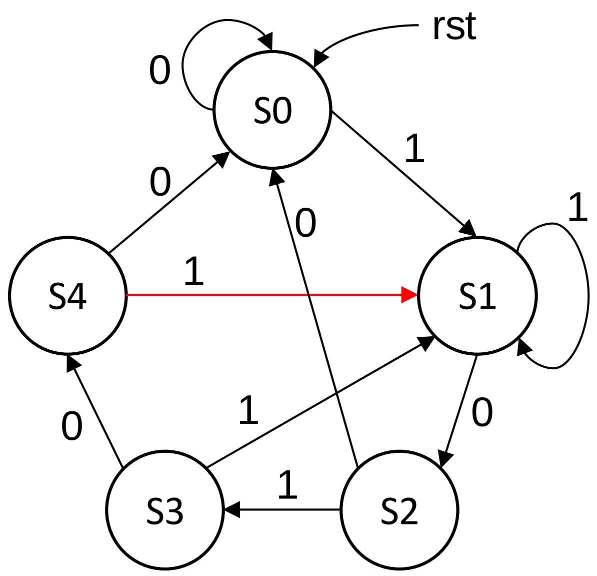
Data	Q2	Q1	Q0	D2	D1	D0
1	0	0	1	0	1	0
0	0	1	0	0	1	1
1	0	1	0	0	1	0

Then, $D1 = data \cdot \overline{Q2} \cdot \overline{Q1} \cdot Q0 + \overline{data} \cdot \overline{Q2} \cdot Q1 \cdot \overline{Q0} + data \cdot \overline{Q2} \cdot Q1 \cdot \overline{Q0}$
$= data \cdot \overline{Q2} \cdot \overline{Q1} \cdot Q0 + \overline{Q2} \cdot Q1 \cdot \overline{Q0}$

ii. Exercise

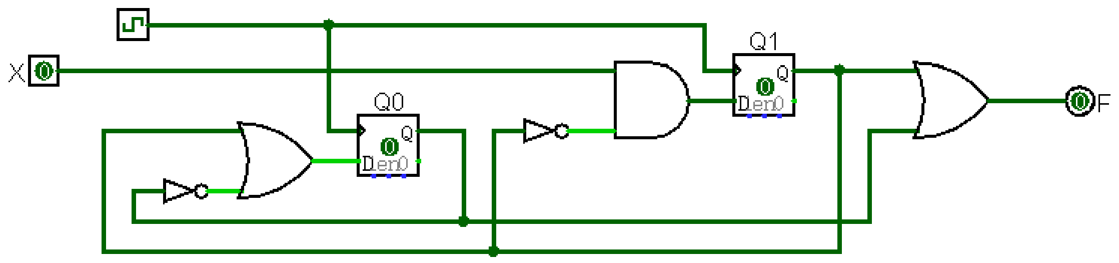
1. On a sequence detector, the input is **data** and the output is **Q**. When **1010** is detected, **Q** outputs **1** for **one cycle**, otherwise **Q** outputs **0**. Give the state machine's substate logic expression and Q's logic expression. Note: **{1010}** is an independent detection and is not stitched with subsequent sequences, that is, **{101010}** is regarded as one successful match.
   Final Answer: Assume that the codes 000, 001, 010, 011, and 100 of the register D2D1D0 correspond to S0 to S4, respectively. Since the Q output width is in units of cycles, Q can only be a function of state, that is, the state machine is a **Moore** type state machine.

输入	现态			次态			
data	Q2	Q1	Q0	D2	D1	D0	Q
0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
1	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0
1	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0
1	0	1	1	0	0	1	0
0	1	0	0	0	0	0	1
1	1	0	0	0	0	1	1



$$D2 = \overline{Q2} \cdot Q1 \cdot Q0 \cdot \overline{data}$$
$$D1 = \overline{Q2} \cdot \overline{Q1} \cdot Q0 \cdot \overline{data} + \overline{Q2} \cdot Q1 \cdot \overline{Q0} \cdot data$$
$$D0 = \overline{Q2} \cdot data + \overline{Q1} \cdot \overline{Q0} \cdot data$$
$$Q = Q2 \cdot \overline{Q1} \cdot \overline{Q0}$$

2. The state machine is shown below. Construct substate logic expressions and output expressions, write state transition and output tables, draw state diagrams, and analyze state machine types.

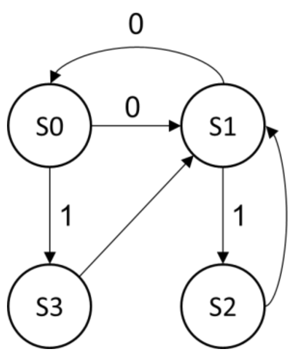


Final Answer:

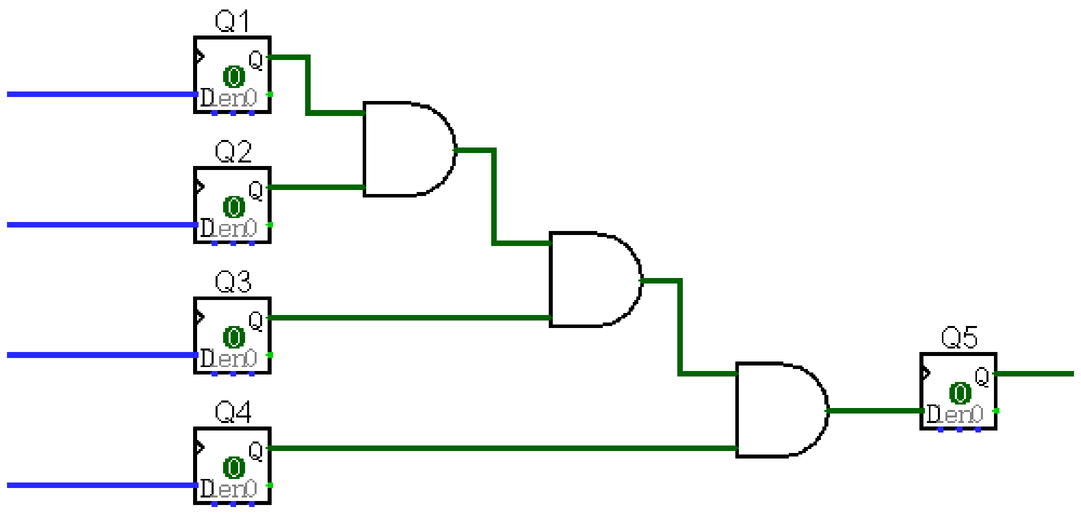
Moore state machine, because F is only related to the status register.

$$Q1 = \overline{Q1} \cdot X$$
$$Q0 = Q1 + \overline{Q0}$$
$$F = Q1 + Q0$$

Q1	Q0	X	$Q1^n$	$Q0^n$	F	现态	X	次态
0	0	0	0	1	0	S0	0	S1
0	0	1	1	1	0	S0	1	S3
0	1	0	0	0	1	S1	0	S0
0	1	1	1	0	1	S1	1	S2
1	0	0	0	1	1	S2	0	S1
1	0	1	0	1	1	S2	1	S1
1	1	0	0	1	1	S3	0	S1
1	1	1	0	1	1	S3	1	S1



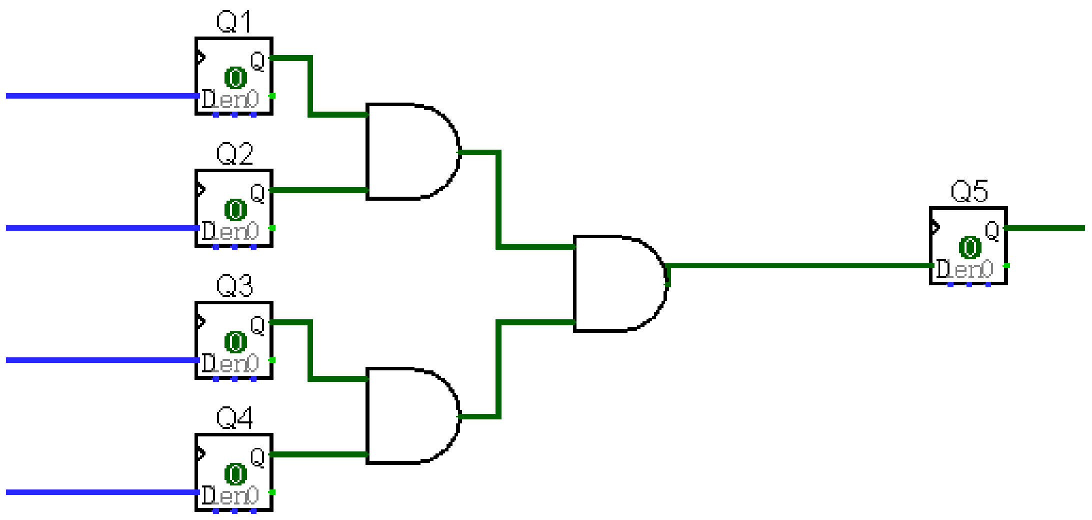
3. For the following circuit, it is assumed that the delay of each AND gate is T, and the connection delay and register inherent delay are all 0.
a) Calculate the critical path delay and the maximum clock frequency of the registers.
b) Restructure the circuit to increase the clock frequency and calculate the increase ratio.



Final Answer:

a) The critical path is a concatenation of 3 AND gates with a delay of 3T. The maximum clock frequency is 1 / 3T.

b) After optimization, it is a 2-layer AND with a delay of 2T. The maximum clock frequency is 1 / 2T, which is 50% higher than the original design frequency.



4. 《数字设计和计算机体系结构》：第 3.20 题。

Final Answer:

1) Since the output is related to the input, this is a Meely type state machine.

2) Two registers Q1 and Q0, the encoding values are: S0 = 0b00; S1 = 0b01; S2 = 0b10

3) Get the following truth table according to the state diagram. The key point is the analysis of S2 to S0 (the green part in the table. Whether it is useful for simplification or not, it must be completed first!)

现态 S1S0	A	B	次态 S1S0	F
00	0	X	00	0

00	1	X	01	0
01	X	0	00	0
01	X	1	10	0
10	1	1	10	1
10	0	0	00	0
10	0	1	00	0
10	1	0	00	0

4) The expression is as follows:

$$S1 = \overline{S1}\cdot S0\cdot B + S1\cdot \overline{S0}\cdot A\cdot B$$
$$S0 = \overline{S1}\cdot \overline{S0}\cdot A$$
$$F = S1\cdot \overline{S0}\cdot A\cdot B$$

5) Function: Check that A and B are input as 1, and then AB is 1 at the same time.

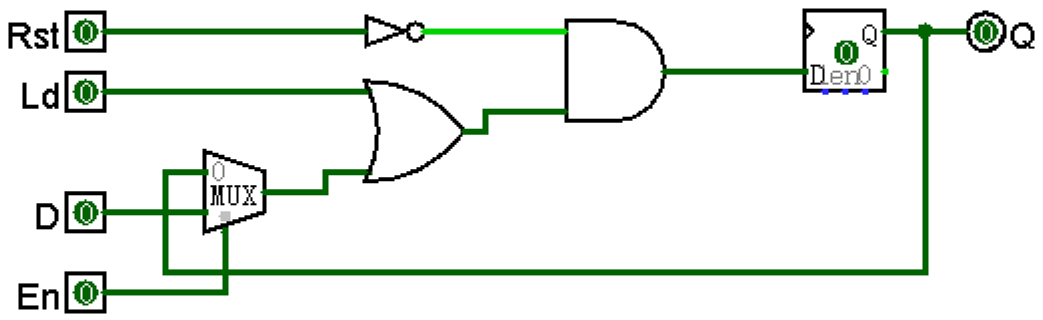
5. Using the D flip-flop as the core, design a support enable (if En is 0, the register value does not change), preset (Ld is 1, the register value is 1), clear (Rst is 1, the register is (Clear) function register. The register input signal is D and the output is Q. All control signals are synchronous control signals. The priority of the three signals is from high to low: Rst, Ld, En.

Final Answer;

1) Because the three control signals are synchronous control signals, the input of the D flip-flop is a function of the three control signals and D.

Rst	Ld	En	D	Q	D 触发器输入
1	X	X	X	X	0
0	1	X	X	X	1
0	0	0	X	Q	Q
0	0	1	0	X	0
0	0	1	1	X	1

2) The circuit structure is shown below



C. Basic Knowledge

i. Basics

- 0 means positive number; 1 means negative number
- Complement code range: [-2^{n-1}, $2^{n-1}-1$]
- To negate, flip the bits and add 1. Example:
  1. If $[x]_{complement} = x_0x_1x_2 \dots x_{n-1}$; then $[-x]_{complement} = \bar{x}_0\bar{x}_1\bar{x}_2 \dots \bar{x}_{n-1} + 1$
  2. +7 = 0b 0000 0111, -7 = 0b 1111 1001
- Represent the same number using more bits: copy MSB
  Example: 0b11 = 0b1111

- High Voltage (V_{dd}) represents 1
- Low Voltage (0 volt or Ground) represents 0
- Let threshold voltage (Vth) decide if a 0 or a 1 (Vth 一般小于 Vdd)
- Our switches: CMOS transistors
- N-channel transistor: open when voltage at Gate is low; close when voltage(Gate) > voltage (Threshold)
- P-channel transistor: open when voltage(Gate) > voltage(Threshold); close when voltage at Gate is low

ii. Exercise

1. Calculate the decimal values corresponding to 222_3, 222_4, 222_5.
   Final Answer: 222_3=26； 222_4=42； 222_5=62
2. Calculate the decimal values of 0b1011101 and 0xB23 respectively (both are treated as unsigned numbers)
   Final Answer: 93； 2851
3. Express the unsigned number is 1001111110_2 in hexadecimal.
   Final Answer: 0x27E
4. In addition to the method of converting decimal to binary, the following two specific cases are shown below. Based on this case, a general method for converting from decimal to N is summarized.

表 III-1 十进制 2007 转换为 5 进制的计算过程

步骤	被除数	商	余数	位序	备注
1	2007	401	2	0	2007 除以 5
2	401	80	1	1	401 除以 5
3	80	16	0	2	80 除以 5
4	16	3	1	3	16 除以 5
5	3	0	3	4	被除数小于除数 5，计算结束

43210 位序

2007_{10} = 31012_5

表 III-2 十进制 2018 转换为 9 进制的计算过程

步骤	被除数	商	余数	位序	备注

1	2018	224	2	0	2018 除以 9
2	224	24	8	1	224 除以 9
3	24	2	6	2	24 除以 9
4	2	0	2	3	被除数小于除数 9，计算结束

3210 位序

$2018_{10} = 2682_9$

Final Answer: Organization cycle: the dividend is divided by the dividend until the quotient is 0; the dividend is adjusted to the last quotient. Sorting the remainders in reverse order is the result.

6. Give a quick estimate of 2^{28}.
   Final Answer: $2^{28}=2^{20}*2^{8}\approx256*10^{6}$
7. Give the 6-bit binary complement representation range.
   Final Answer: {-32， +31}
8. Convert the following decimal numbers to 6-bit complement binary and complete the calculation, and indicate if there is an overflow in the result.
   ①16+15 ② 16+18 ③16-8 ④-16-16 ⑤-24-13
   Final Answer: 1） 011111； 2） Overflow； 3） 001000； 4） 100000； 5） Overflow
9. After the execution of the following code, please use 32-bit binary complement to represent the values of c, s, and us, respectively.

```
char           c ;
short          s ;
unsigned short us ;

c = -1 ;
s = c ;
us = (unsigned short)c ;
```

Final Answer: c： 0xFFFF_FFFF； s： 0xFFFF_FFFF； us=0x0000_FFFF

D. Instructions and Assembly

i. Basics

- Mips Register: 0~31, **Below are the significant ones**

$1	->	$at	
$2-$3	->	$v0-$v1	
$4-$7	->	$a0-$a3	VOLATILE
$8-$15	->	$t0-$t7	VOLATILE
$16-$23	->	$s0-$s7	**PROTECTED**
$24-$25	->	$t8-$t9	VOLATILE
$29	->	$sp	
$31	->	$ra	**PROTECTED**

- MIPS Instruction types:
  Opcode|Funct -> 6 bits

Imm	->	16 bits
RD|RS|RT	->	5 bits
Shamt	->	5 bits

To get the hex code of an instruction, divide all the instruction binary into 4

Example:

add $8, $9, $10

Opcode	rs	rt	rd	shamt	funct
31:26 = 0	25:21 = 9	20:16 = 10	15:11 = 8	10:6 = 0	5:0 = 32
-> 000000	01001	01010	01000	00000	100000

```
-> 0000_0001_0010_1010_0100_0000_0010_0000
-> 0    1    2    A    4    0    2    0
-> 0x012A_4020
```

- Often used instructions:

  Any part of the 32 that is not mentioned is ALWAYS 0!!!

  - I-TYPE:

Instr	**rt, rs, imm**
ADDI:	OPCODE 8
ADDIU:	OPCODE 9
ANDI:	OPCODE 12
BEQ:	OPCODE 4
BNE:	OPCODE 5
ORI:	OPCODE 13
XORI:	OPCODE 14

Instr	**rt, imm**
LUI:	OPCODE 15

Instr	**rt, imm(rs)**
LB:	OPCODE 32
LBU:	OPCODE 36
LH:	OPCODE 33
LHU:	OPCODE 37
LW:	OPCODE 35
SB:	OPCODE 40
SH:	OPCODE 41
SW:	OPCODE 43

  - J-TYPE:

Instr	**label**
J:	OPCODE 2
JAL:	OPCODE 3

  - R-TYPE: (OPCODE 0)

Instr	**rd, rs, rt**
ADD:	FUNCT 32
ADDU:	FUNCT 33
AND:	FUNCT 36
NOR:	FUNCT 39
OR:	FUNCT 37
SLLV:	FUNCT 4
SLT:	FUNCT 42
SRAV:	FUNCT 7
SRLV:	FUNCT 6
SUB:	FUNCT 34
SUBU:	FUNCT 35
XOR:	FUNCT 38

Instr	**rd, rt, sa**
SLL:	FUNCT 0
SRA:	FUNCT 3
SRL:	FUNCT 2

Instr	**rs, rt**
DIV:	FUNCT 26
DIVU:	FUNCT 27
MULT:	FUNCT 24
MULTU:	FUNCT 25

Instr	**rd, rs**
JALR:	FUNCT 9

Instr	**rs**
JR:	FUNCT 8

Instr	**rd**
MFHI:	FUNCT 16
MFLO:	FUNCT 18

move;
la; li

- In Store/Load word instructions, for the offset, just use an immediate which is power of 4, byte is power of 1

Example:
```
lw $t0, 12($s3)  -> $t0 = A[3]
sw $t0, 40($s3)  -> A[10] = $t0
```

```
               B3_B2_B1_B0
*($s0) = 0x00_00_01_80
```

lb $s1, 1($s0) //Since we want to get the value of index 1, then we'll get the value of B1 (01), and then the rest of the 24 bits are sign-extended (the sign depends on the highest bit), so

$s1 = 0x00_00_00_01

lb $s2, 0($s0) //$s2 = 0xFF_FF_FF_80
sb $s2, 2($s0) //*($s0) = 0x00_80_01_80

- For pointers (e.g *p), can use memory of $s0. So for example lb$t0, 0($s0) = $t0 = *p
- In div, the LO is the result of division while HI is result of mod
- The smallest negative number: 0x80000000 = -2147483648
- Shifting instruction example:
  addi $t0,$0 ,-256 # $t0=0xFFFFFF00 = -256
  sll $s0,$t0,3 # $s0=0xFFFFF800 = -2048, the '3' means 2^3
- Lui/ori example:

  ```
  addi $t0, $t0, 0xABAB_CDCD
  EQUALS TO:
  lui $at, 0xABAB           //Load Upper imm
  ori $at, $at, 0xCDCD      //Load Lower imm
  add $t0, $t0, $at         //Assign value
  ```
- When using jal, the address that is saved to $ra is PC+4 and not PC
- Beq/bne can only jump to a total of <50 instructions
  If the comparison result is True: PC = PC + 4
  False: PC = (PC + 4) + (immediate * 4)
  Example:

  ```
  1    Loop:  beq    $9, $0, End
  2           addu   $8, $8, $10
  3           addiu  $9, $9, -1
  4           j      Loop
  5    End:
  ```
  Opcode = 4; rs = 9; rt = 0, imm = 3 (because End is 3 instructions away from **the instruction after beq**)
- J Instruction jump range is 256MB, because (range = {**X**000_0000, **X**FFF_FFFF})
- JR can jump within 4GB range of addr

ii. Exercise

1. Please analyze the rationality of 32 MIPS registers from the perspective of instruction encoding format. Tip: It mainly analyzes the negative impact on the instruction encoding if the number of registers is more than 32.
   Final Answer:
   If there are more than 32, the number of coding bits is at least 6 bits.
   1) R-type shift instruction: The number of shift bits is 4 bits, and a single instruction cannot be arbitrarily shifted within 32 bits.
   2) Type I calculation: the immediate part is reduced to 14 digits. Expansion to 32 bits requires at least 2 instructions.
   3) Type I beq: The branch address range is greatly reduced.

2. Analyze the range of the entry address of the called function of jal in units of instructions.
   Final Answer: If 4G is divided into 16 256MB, you can jump arbitrarily in the 256MB section where jal is located.
3. Based on the characteristics of the C language, analyze the rationality of the imm field in the beq instruction format.
   Final Answer: The imm of beq has 16 bits, which is equivalent to the range of 64K instructions. It means that with beq as the benchmark, the range that can be jumped up and down is {-32K, + 32K}. A C statement is larger than about 10 instructions. This means that the size of the if-else block is 6.4K C statements. From the perspective of program design rationality, there should not be such a large block of statements. So the range of imm is enough.
4. Please translate the following C code into assembly code. Assuming $s0 stores the variable i, only $s0 ~ $s3 registers available.

```
do {
   Loop Content;
while ( 0<i && i<100 ) ;
```

   Final Answer:

```
Loop :
        Loop Content
        blez $s0, Loop_End      // Quit when i ⩽ 0
        slt $s0, $s1, 100       // When i ⩾ 0, $s1 is 0
        beq $s1, $0, Loop_End   //
        j Loop
Loop_End :
```

5. Use beq and bne and branch-independent instructions to complete the function of the following statement. Tip: To prevent overflow.
   slt $s0, $s1, $s2
   Final Answer:

```
        s1<s2:  s0=1
        s1==s2:  s0=0
        s1>s2:  s0=0

        sub $t0, $s1, $s2
        beq $t0, $0, SET0     // s1==s2
        srl $t0, $t0, 31      // Keep only sign bit
        beq $t0, $0, SET0     // Sign bit is 0:  S1>S2
        addi $s0, $0, 1       // Sign bit is 1:  S1<S2
        j END
SET0:
        addi $s0, $0, 0
END:
```

6. The programmer wrote the following assembler to complete the 1000-byte copy task. Assume that before the loop starts, $s0 and $s1 point to the first address of the source string and the first address of the destination string, respectively.

```
LOOP:   lb $t0, 0($s0)
        beq $t0, $0, TAIL
        sb $t0, 0($s1)
        addi $s0, $s0, 1
        addi $s1, $s1, 1
        j LOOP
TAIL:
```

1) Calculate the total number of instructions executed by the above code.
2) Please maximize the above code. Optimize code using only the instructions taught in this chapter. The optimized code still uses a loop structure, and there can be only one copy operation per loop.
3) Calculate the total number of instructions executed by the optimized code.

Final Answer:
1) For 1000 bytes, a 1-byte terminator (that is, 0) must be added, so the total number of cycles is 1001 times. So the total number of instructions executed = 6 * 1001 = 6006
2) Loop 250 times for copying, then S1 points to the [word] where the terminator is located. To do this, the highest byte in the word needs to be written (because the highest byte is the terminator)

```
        addi $t1, $0, 250
Loop:
        lw $t0, 0($s0)
        sw $t0, 0($s1)
        addi $s0, $s0, 4
        addi $s0, $s1, 4
        addi $t1, $t1, -1
        bne $t1, 0, LOOP
        sb $0, -3(s1)
```

3) Total instructions = 6 * 250 + 1 = 1501

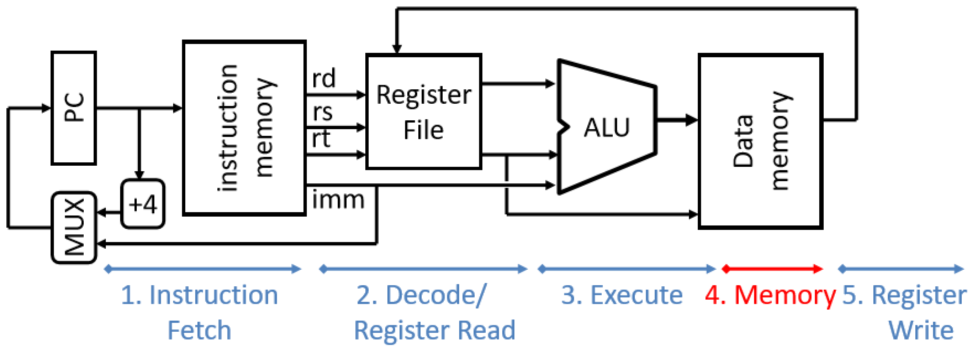
7. Write a recursive function called fib (n) to calculate the nth Fibonacci number. The calculation formula of Fibonacci sequence is: f (n + 2) = f (n + 1) + f (n), f (1) = f (2) = 1.
1) Test the program with the MARS simulator.
2) If the stack space is 4KB, please estimate the limit of the number of recursive calls.

Final Answer:
1) (Ignore)
2) Assume that the total number of registers that the function needs to put on the stack is n (including PC and general-purpose registers, etc.), so the stack capacity requirement is 4n bytes. The limit of the number of calls is 4K / 4n = 1K / n.

E. CPU

i. Basics

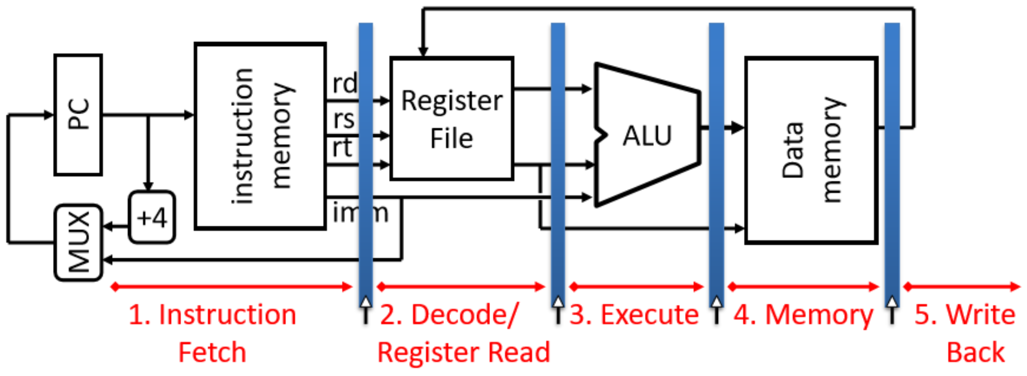
- CPU Time = Instructions x CPI x Clock Cycle Time
- In case wanna memorize the Datapath of Single-cycle CPU



When counting action time example:
100ps for reg read/write; 200ps for the rest
Lw -> F (200ps) -> D (100ps) -> E (200ps) -> M (200ps) -> W (100ps)
Sw -> F (200ps) -> D (100ps) -> E (200ps) -> M (200ps) -> W (None)

- In Ctrl there are: ExtOp; ALUsrc; ALUctr; nPC_sel; MemWrite; MemtoReg; RegDst; RegWrite;

- Pipeline CPU

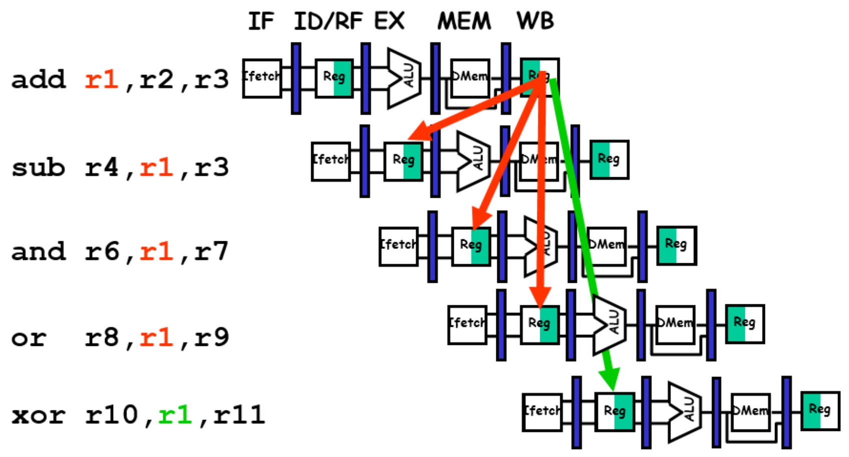


5 Stages:

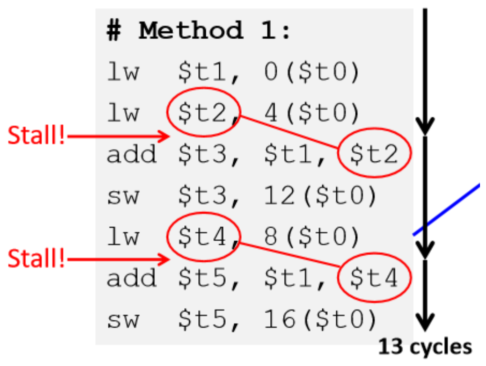
1. IF: Instruction Fetch, Increment PC
2. ID: Instruction Decode, Read Register
3. EX: Execution (ALU) -> Load/Store (Calculate Address), Others (Perform Calculation)
4. MEM: Load/Store
5. WB: Write to Register

Hazard:

1. Structural Hazard: Read/Write Memory/Regs at the same clock cycle
2. Data Risks:
   - Data dependency between instructions
   - Forwarding
   - Load Delay Slot



   The r1 in red means it's still reading the old data, and the green one means new. Because pipeline will be delayed by clock cycles, new data couldn't be updated right away, therefore there will be risks.



```
# Method 1:
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

3. Control Hazard: Stall on every branch/jump until we have the new PC value
   Always give PC+8 instead of PC+4

ii. Exercise

1. In order to forward the DM results to the ALU, this chapter adopts the idea shown in Figure 6-16 (a): from the last-stage pipeline register MEM / WB to the ALU. However, this design must suspend one clock cycle while executing the following instruction sequence. The designer believes that the problem can be solved by adopting the idea shown in Figure 6-16 (b): forwarding directly from DM to ALU. Although Figure 6-16 (b) can solve the above problem, it has reduced the pipeline clock frequency. Please analyze the specific reason (assuming that the delays of IM read, RF read, ALU, and DM read are L).
   lw $1, xxx
   add yyy, $1, zzz

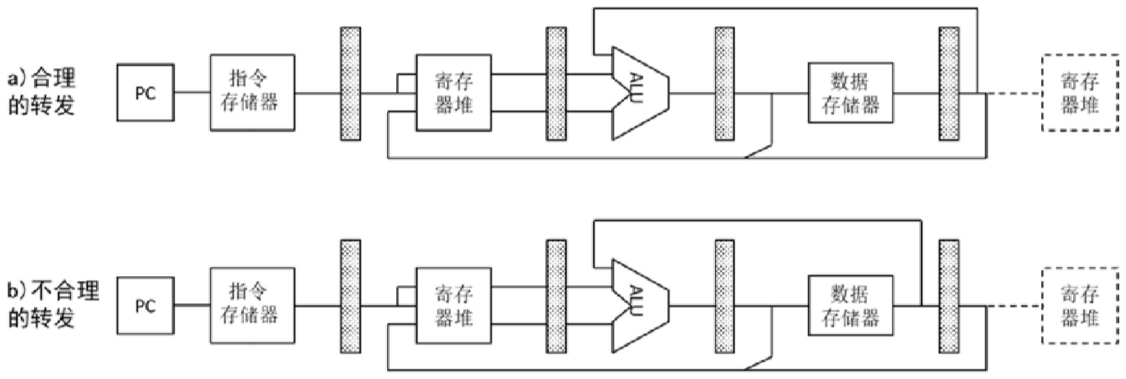


图 6-16 DM 转发的两种思路

Final Answer: Due to the feedback from the DM to the ALU, there is a combination logic from the EX / MEM register to itself, namely DM + ALU, with a delay of 2L. The pipeline frequency becomes 1 / 2L, so the pipeline performance of Figure b is 50% of that of Figure a.

2. Inserting registers in the pipeline can increase the clock frequency. However, as the number of stages increases, pipeline performance improvement will encounter bottlenecks. First, the timing overhead of the registers themselves (such as register setup time and hold time, etc.) has an increasing impact on performance improvement. Secondly, with the increase of the number of pipeline stages, branch risk will lead to more serious problems of pipeline emptying. Third, the pauses caused by data adventures will also increase.

Assumption: The 5-level pipeline CPI is 1.2, and for each additional level, the CPI increases by 0.1; the single-cycle CPU critical path delay is 800ps, and the register's own timing overhead is 50ps.

1) Establish the calculation formula of CPI and pipeline stage number N (N⩾5).

2) Establish a formula for calculating the clock cycle delay Tc and the number of pipeline stages N.

3) Give a calculation formula for the execution time of an instruction.

4) Please indicate how much N is the best pipeline performance.

5) Please indicate how much N improves the performance of the pipeline.

Final Answer:

1) $CPI = 1.2 + 0.1(N-5) = 1.2 + 0.1N - 0.5 = 0.1N + 0.7$

2) $T_c = \frac{800}{N} + 50$

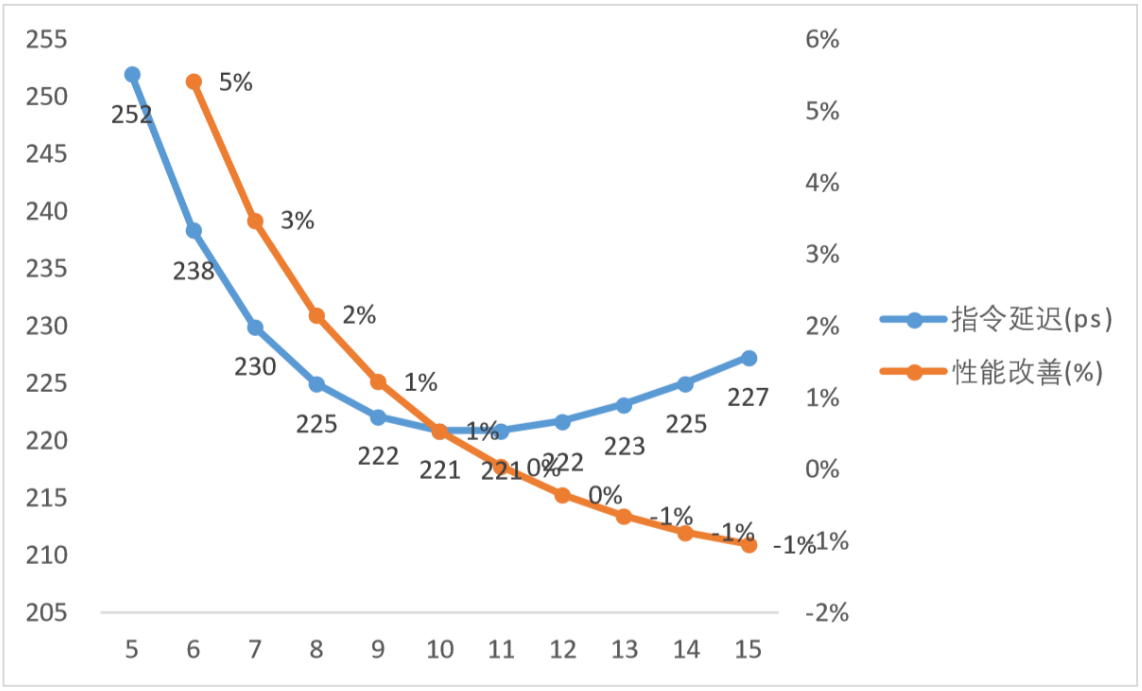
3) Execution time of an instruction = $CPI \times T_c = (0.1N + 0.7) \times \left(\frac{800}{N} + 50\right)$

$$= 80 + \frac{560}{N} + 5N + 35$$

$$= 115 + \frac{560}{N} + 5N$$

4) N = 10, Best performance

5) N = 6, The most significant performance improvement



3. The designer transformed the single-cycle data path into a three-stage pipeline as shown in Figure 6-17. It is assumed that RF does not support internal forwarding.

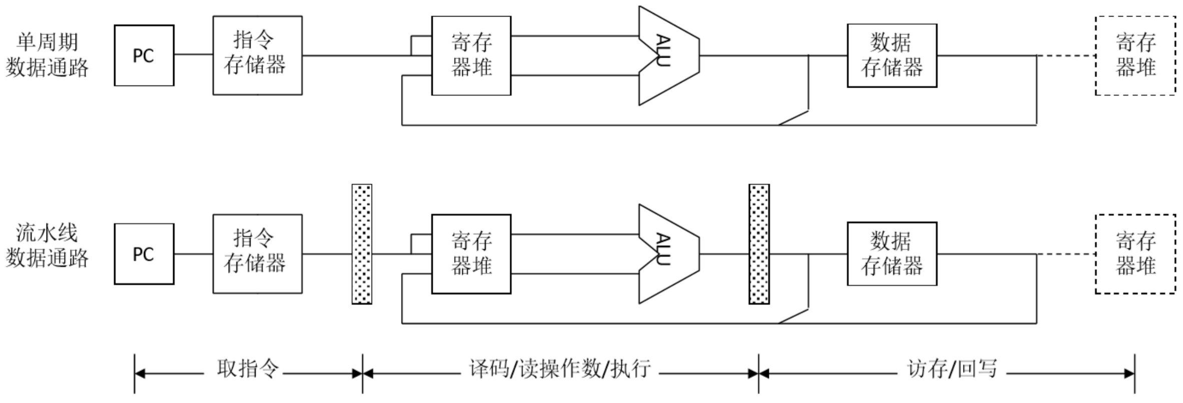


图 6-17 改造单周期数据通路为 3 级流水线

1) Is it still possible that the pipeline needs to be emptied due to beq? If so, how many instructions will be cleared?

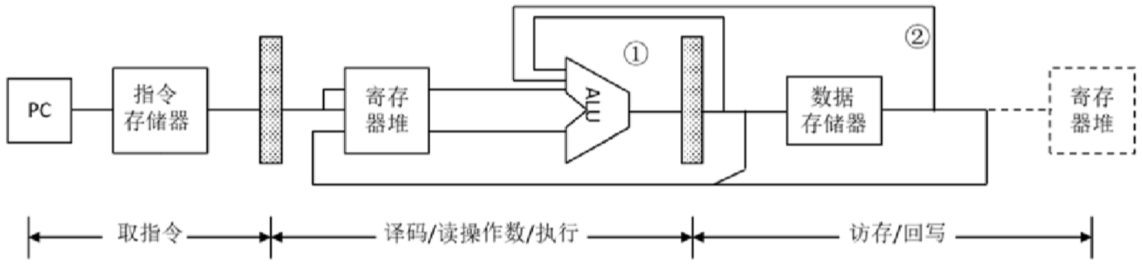
2) Assuming that the instruction set is only {lw, add}, please take the rs register as an example, and add a bypass to deal with all data risks. It is enough to give the idea, without discussing the MUX and its control.

3) For the instruction set in question 2, can all the data risks related to the rs register be eliminated? why?

Final Answer:

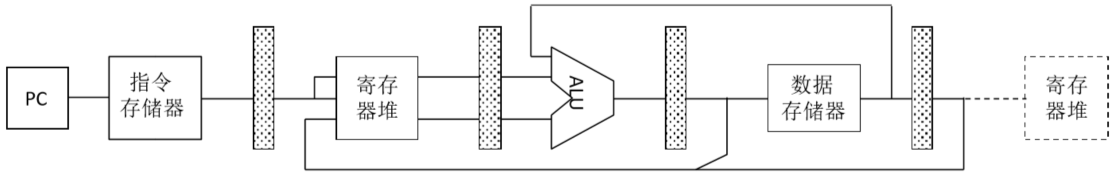
1) Yes. The comparison and judgment circuit of beq can only move forward to the decode / read operand / execute stage. This is completely consistent with the process

described in this chapter. The instruction that may be cleared is still the one following beq, so there is only one.
2) Forward data from ALU and DM to A-side of ALU respectively. Note that because RF has no internal forwarding, path 2 must be present.

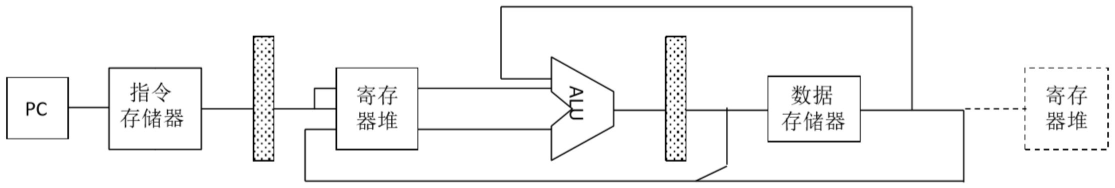


3) can. Because for {lw, add}, rs is only used on the A side of the ALU. No matter which instruction currently requires the use of ALU, there can be only one instruction in front of it at the last level. Regardless of which instruction is at the last level, the results have already been generated, so the data risk can be eliminated by bypassing forwarding.

4. It is also the forwarding from DM to ALU. The design of Figure 6-18 (a) will degrade the performance of the 5-stage pipeline. Please analyze the design of Figure 6-18 (b). Will the performance of the 3-stage pipeline decrease? It is assumed that the delay of IM read, RF read, ALU, DM read is L, and all controller delays and MUX delays are ignored.



5级流水线：不合理的转发



3级流水线：合理的转发

图 6-18 从 DM 向 ALU 的转发

Final Answer:
1) In a 3-stage pipeline, the middle delay is already the worst 2L.
2) The combination logic of the forwarding circuit is DM readout + ALU, which is also 2L.
3) This shows that forwarding does not increase the worst-case delay, so it will not cause performance degradation.

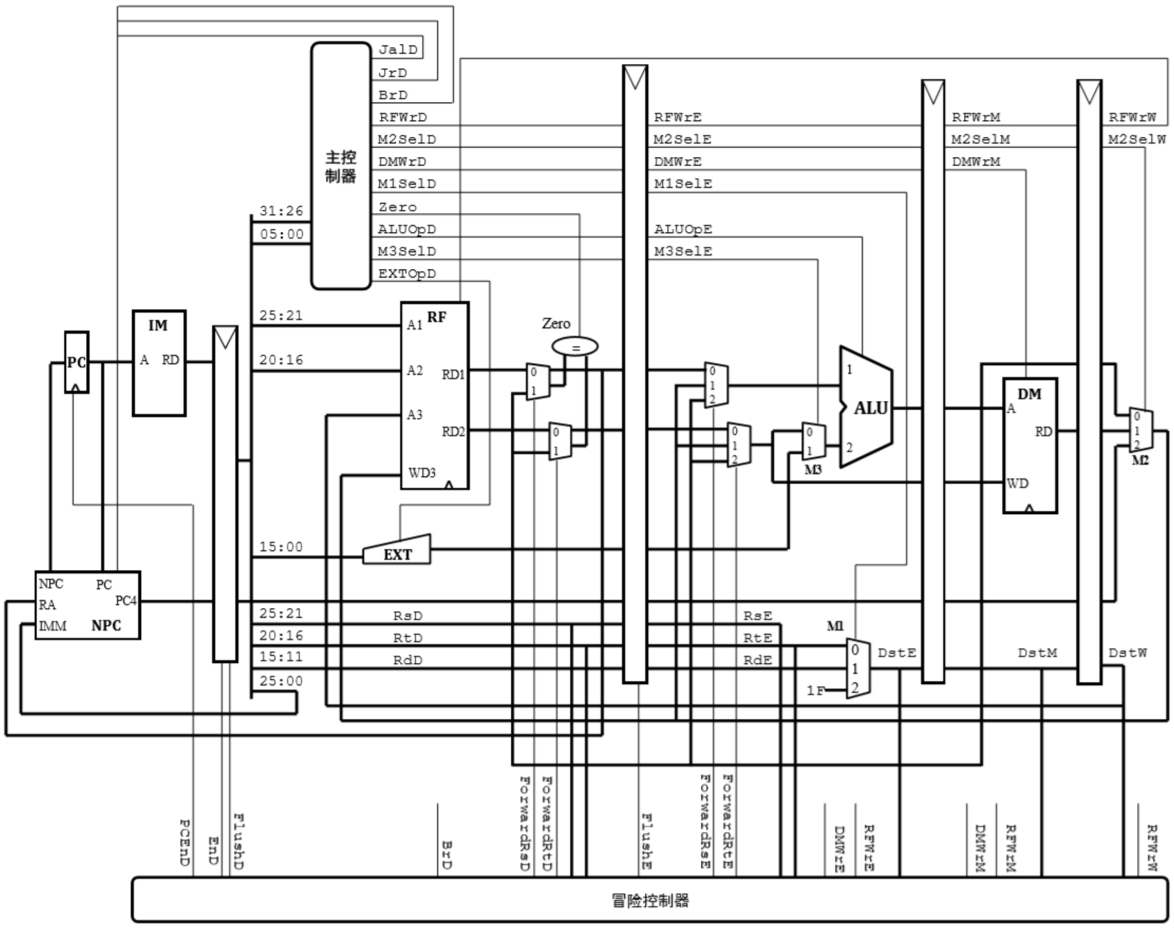
5. The pipeline CPU shown in the figure below executes the following instruction sequence

I1	lw $1, 0($2)
I2	addi $1, $1, $1
I3	sw $1, 0($2)
I4	lw $1, 4($2)

I5 sw $1, 8($2)

1) Analyze how many clock cycles the pipeline needs to suspend during the execution of the above instructions.

2) Can I increase forwarding to improve pipeline performance? If possible, please briefly describe the design ideas.



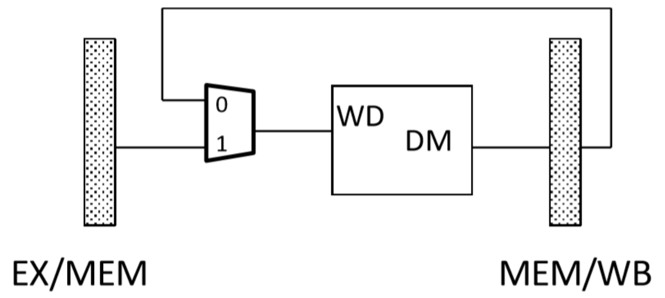
Final Answer:

1) There is a risk of data between instruction 1 and instruction 2 at $ 1. If the forwarding circuit exists, it must still be suspended for 1 clock cycle. The instruction sequence is adjusted to:

```
1  lw $1, 0($2)
2  nop
3  addi $1, $1, $1
4  sw $1, 0($2)
5  lw $1, 4($2)
6  sw $1, 8($2)
```

There is a risk of data between original instruction 4 (now instruction 5) and original instruction 5 (now instruction 6). But the original design does not have a forwarding circuit, so sw must be frozen at IF / ID until lw enters MEM / WB. At this time, through RF internal forwarding, sw can get the correct $ 1. The number of pause periods is two. Therefore, the total pause time of the above code is 3 clock cycles.

2) The two pause periods in the previous question are caused by solving the data adventure between lw-sw, so we need to add a WD forwarded from MEM / WB to DM.



6. The pipeline CPU shown in Figure 5 executes a program, and its instruction distribution is as follows: load accounts for 15%, store is 10%, branch instructions are 10%, and R-type calculation instructions are 65%. Assume: load-R causes a pause probability of 30%; load-store causes a pause probability of 5%; branch instruction prediction success rate is 75%. The calculation pipeline executes the CPI of the program.
   Final Answer:
   5) load-R: When there is no data related, the CPI of load is 1. If there is data related, it needs to suspend for 1 clock cycle, and its CPI is 2.
   $$CPI_{load-R} = 1 \times (1-30\%) + 2 \times 30\% = 1.3$$
   6) load-store: When there is no data related to data, the CPI of load is 1. If there is data correlation, it needs to suspend for 2 clock cycles, and its CPI is 3.
   $$CPI_{load-store} = 1 \times (1-5\%) + 3 \times 5\% = 1.1$$
   7) store: There is no data correlation in the question, so the $\text{CPI}_{\text{store}}$ is 1
   8) Branch: The prediction was successful. The CPI of the branch is 1. If the prediction fails, it needs to suspend for 1 clock cycle, and the branch CPI is 2.
   $$CPI_{branch} = 1 \times 75\% + 2 \times (1-75\%) = 1.25$$
   9) R-Type: CPI is 1
   $$\begin{aligned} CPI = {} & CPI_{load-R} \times 30\% + CPI_{load-store} \times 5\% + CPI_{store} \times 10\% \\ & + CPI_{branch} \times 10\% + CPI_{R-Type} \times 65\% \\ = {} & 1.3 \times 30\% + 1.1 \times 5\% + 1 \times 10\% + 1.25 \times 10\% + 1 \times 65\% \\ = {} & 1.32 \end{aligned}$$

7. As shown in Figure 6-19, a MIPS standard level 5 pipeline only supports forwarding from level M to level D (note: there is no internal forwarding in the register file). A programmer wrote the following MIPS code, please answer the following questions.

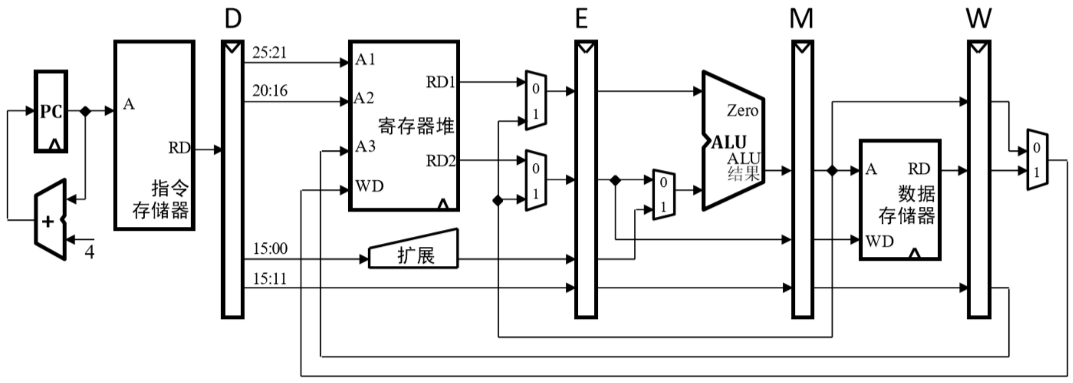


图 6-19 仅支持 M 级向 D 级转发的 5 级流水线

I1: lw $1, 0($2)
I2: sw $1, 0($1)
I3: add $3, $2, $2
I4: sub $4, $1, $3
I5: or $5, $5, $6

1) Please point out that all the data existing when the above instruction fragment is executed in the above pipeline.
2) Please optimize the above instruction fragments by adjusting the instruction sequence to minimize the pause.
3) For the pre-optimized and post-optimized instruction fragments, the execution time of the pipeline is given, and the reason is explained. Example: For 2 risk-free instruction fragments, the pipeline execution time is 6 clock cycles.

Final Answer:

1) lw-sw is correlated twice at $ 1; lw-sub is correlated at $ 1; add-sub is correlated at $ 3.
2) I1: lw $1, 0($2)
   I3: add $3, $2, $2
   I5: or $5, $5, $6
   I2: sw $1, 0($1)
   I4: sub $4, $1, $3
3) Before optimization: 13 cycles. Because there is only forwarding between W and D, two NOPs must be inserted between lw-sw and add ~ sub. The total number of instructions changed from 5 to 9, so the pipeline needs a total of 9 + (5-1) = 13 clock cycles.

   After optimization: 9 cycles After optimization, the data correlation of lw ~ sw and add ~ sub are all resolved through forwarding, so it is not necessary to insert NOP, so the execution time = 5 + (5-1) = 9 clock cycles

F. Storage/Memory

i. Basics

- **DRAM Chips = Main Memory Capacity / DRAM Capacity**
  Internal address = $\log_2$(DRAM Capacity (take the Megabyte))
- Caches use static RAM (SRAM)
  - Fast
  - Low Density, High Power, Expensive
- Main Memory uses dynamic Ram (DRAM)
  - Slow
  - High Density, Low Power, Cheaper
- TIO Address Breakdown

  Tag : Index : Offset

  O bits = 2^0 bytes/block = 2^{O-2} words/block
  = **$\log_2$({Block Size} (in Bytes))**

  I bits = 2^I rows in cache = cache size / block size
  = **$\log_2$({Cache size / Block Size})**

  T = A – I – O

  B = 8×2^O bits

  A = $\log_2$(Address Space)

  Total bits in cache = # rows x (B + T + 1)
  = $2^I \times (8 \times 2^O + T + 1)$ bits

Cache 块数	**= Cache size/Block size**
Cache 组数	**= Cache 块数 / 组相联**
主存块数	**= 主存容量 / Block Size**
主存组数	**= Cache 组数**
主存每组块数	**= 主存块数 / 主存组数**
主存地址	**= 组内块地址（T） + 组地址（I） + 块内地址（O）**

Example:
Address space of 64B, block size of 1 word, cache size of 4 words

Answer: TIO Breakdown:
- 1 word = 4 bytes, so O = $\log_2(4) = 2$
- Cache size / block size = 4, so I = $\log_2(4) = 2$
- A = $\log_2(64)$ = 6 bits, so T = 6 – 2 – 2 = 2
- Bits in cache = $2^2 \times (8 \times 2^2 + 2 + 1)$ = 140 bits

Example 2:

主存容量 1M 字节，4 路组相联（每组包含 4 个 Block）Cache 容量 16K 字节，Block 大小 256 字节

Cache 分多少组？每组包含多少块？

Cache 的 Tag 需要多少位？

Answer:

Block=256B -> O = 8 位

Cache 组数 = $2^{14} \div (2^8 \times 2^2)$ = 2^4 = 16 组 -> I = 4 位

主存每组块数 = $2^{20} \div (2^8 \times 2^4)$ = 2^8 = 256 块/组

主存地址：20 位，其中高 8 位为组内块地址，中间 4 位为组地址，低 8 位为块内地址

Cache 的 Tag 应该为 8 位。

- Cache Performance
  T_m = 主存储器的访问周期
  T_c = Cache 的访问周期
  H = Cache 命中率
  $\boldsymbol{T = (T_c \times H) + (T_m \times (1 - H))}$
  S_p = 加速比
  $$\boldsymbol{S_p = \frac{T_m}{T} = \frac{T_m}{H \times T_c + (1 - H) \times T_m} = \frac{1}{(1 - H) + H \times {}^{T_c}/_{T_m}}}$$
- Sources of Cache Misses:
  - Compulsory
  - Capacity
  - Conflict
- **CPI$_{\text{stall}}$ = CPI$_{\text{base}}$ + Average Memory – Stall Cycles**
- **Memory-stall Cycles = (Accesses/Instruction) x MR x MP**

Example:
CPI$_{\text{base}}$ of 1, a 100 cycle MP, 36% load/store instructions, and 2% I\$ and 4% D\$ MRs
– How many times per instruction do we access the I\$? The D\$?
– MP is assumed the same for both I\$ and D\$
– Memory-stall cycles will be sum of stall cycles for both I\$ and D\$

Answer:

– Memory-stall cycles = (100% × 2% + 36% × 4%) × 100 = 3.44

– $CPI_{stall} = 1 + 3.44 = 4.44$

- Average Memory Access Time (AMAT):
  AMAT = Hit time + Miss rate x Miss Penalty
  = HT + MR x MP

  Example:
  200ps clock, MP of 50 clock cycles, MR of 0.02 misses/instruction, and HT of 1 clock cycle

  Answer: AMAT = 1 + 0.02 x 50 = 2 clock cycles = 400ps
- Multilevel Cache AMAT
  AMAT = L1 HT + L1 MR x L1 MP
  L1 MP = L2 HT + L2 MR x L2 MP //L1 depends on L2, if more level, then continue

  Example: 1 cycle L1 HT, 2% L1 MR, 5 cycle L2 HT, 5% L2 MR, 100 cycle main memory HT
  Answer:
  - Without L2$: $AMAT_1 = 1 + 0.02 \times 100 = 3$
  - With L2$: $AMAT_2 = 1 + 0.02 \times (5 + 0.05 \times 100) = 1.2$
- Local / Global Miss Rates
  For 2-level cache: MR_{global} = L1 MR x L2 MR
  AMAT = L1 HT + L1 MR x (L2 HT + L2 MR x L2 MP)
  = L1 HT + L1 MR x L2 HT + MR_{global} x L2 MP
  $CPI_{stall} = CPI_{base}$ + (Accesses/Instruction) x L1 MR x (L1 MP + L2 MR × L2 MP)
  $CPI_{stall} = CPI_{base}$ + (Accesses/Instruction) (L1 MR x L1 MP + MR_{global} × L2 MP)

  Example:
  - CPI_{base} of 2
  - 100 cycle miss penalty to main memory
  - 25 cycle miss penalty to unified L2$
  - 36% of instructions are load/stores
  - 2% L1 I$ miss rate; 4% L1 D$ miss rate
  - 0.5% global U(nified)L2$ miss rate

  What is CPI_{stall} with and without the L2$?

  Answer:

  - Without L2$: $CPI_{stall} = 2 + 1 \times 0.02 \times 100 + 0.36 \times 0.04 \times 100 = 5.44$
  - With L2$: $CPI_{stall} = 2 + 1 \times .02 \times 25 + .36 \times .04 \times 25$
    $+ 1 \times .005 \times 100 + .36 \times .005 \times 100$
    $= 3.54$

- Cost of Polling:

Example: 1 GHz clock, 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning)
Determine % of processor time for polling:

- Mouse: Polled 30 times/sec so as not to miss user movement
- Floppy disk: Transferred data in 2-Byte units with data rate of 50 KB/sec. No data transfer can be missed.
- Hard disk: Transfers data in 16-Byte chunks and can transfer at 16 MB/second. Again, no transfer can be missed.

Answer:

- Mouse polling:
  Time taken: 30 [polls/s] × 400 [clocks/poll] = 12K [clocks/s]
  % Time: 1.2×104 [clocks/s] / 109 [clocks/s] = 0.0012%
  Polling mouse little impact on processor
- Disk polling:
  Freq: 16 [MB/s] / 16 [B/poll] = 1M [polls/s]
  Time taken: 1M [polls/s] × 400 [clocks/poll] = 400M [clocks/s]
  % Time: 4×108 [clocks/s] / 109 [clocks/s] = 40%
  Unacceptable!

- Virtual Memory
  TLB:

Valid	Dirty	Ref	Access Rights	TLB Tag	PPN
X	X	X	XX		

- *Valid* and *Access Rights:* Same usage as previously discussed for page tables
- *Dirty:* Basically always use write-back, so indicates whether or not to write page to disk when replaced
- *Ref:* Used to implement LRU
  – Set when page is accessed, cleared periodically by OS
  – If Ref = 1, then page was referenced recently
- *TLB Tag:* VPN mod (# TLB entries)

虚（逻辑）地址：	**虚页号 + 页内偏移**
实（物理）地址：	**实页号 + 页内偏移**
页内偏移：	**$\log_2$(页大小)**
每个程序虚拟空间最多可有：	**$2^{\text{虚页号}}$**
每个页表项：	**有效位 + 实页号**
每个页表所占空间：	**$2^{\text{虚页号}}$ x 页表项**

TLB 组数：	**TLB 总表项数/组相联**
TLB 每个表项：	**Tag + 每个页表项**
TLB 容量：	**TLB 每个表项 x 表项数**

Example:
某计算机虚拟地址 32 位，物理内存 128MB，页大小 4KB。
（1）程序虚拟空间最多可有多少页？
（2）页表项共有多少位？
（3）每个页表占多少内存空间？

Answer:
虚地址 32 位：20 位 + 12 位
实地址 27 位：15 位 + 12 位
每个程序虚拟空间最多可有：2^{20} 个虚页
每个页表项：1 位 + 15 位 = 16 位
每个页表所占空间：$2^{20} \times 16 = 16Mb = 2MB$

Example 2:
假定页式虚拟存储系统按字节编址，逻辑地址 36 位，页大小 16KB，物理地址 32 位，页表中包括有效位和修改位各 1 位，使用位和存期方式位各 2 位，且所有虚拟页都在使用中。请问：
（1） 每个进程的页表大小为多少？
（2） 如果所使用的快表（TLB）总表项数为 256 项，且采用 2 路组相联 Cache 实现，则快表大小至少为多少？

Answer:

(1) 页面大小：16KB = 2^{14}，页内偏移 14 位
    虚地址 36 位：虚页号 = 36 – 14 = 22 位
    实地址 32 位：实页号 = 32 – 14 = 18 位
    每个进程最多可有：2^{22} 个虚页
    每个页表项：1 + 1 + 2 + 2 + 18 = 24 位
    每个页表所占空间：2^{22} x 24 = 12MB
(2) TLB: 256 个表项，2 路组相联，以供有 128 组
    22 位虚页号：7 位组地址，15 位 Tag
    TLB 每个表项：15 + 24 = 39 位
    TLB 容量：39 x 256 = 9984 位 = 1248 字节

Example 3:
How many bits wide are the following?

16 KiB pages
40-bit virtual addresses
64 GiB physical memory
2-way set associative TLB with 512 entries

ii. Exercise

1. The ideal CPI of a standard pipeline CPU is 1, and the load and store instructions in the program fragments it executes account for 30%, and the remaining instructions are all R-type instructions. The main memory delay is 50 clock cycles. L1 level cache performance is: the hit time is 1 clock cycle, and the miss rate is 2%. L2 cache performance is: the hit time is 10 clock cycles, and the miss rate is 5%. The actual CPI of the pipeline is calculated separately for the following situations: no cache; only L1 cache; L2 cache.

   Final Answer:

   1) Because there is no cache, the missing rate of instructions and data is 100%;
   Pause Cycles = instruction missing rate × instruction missing cost + data missing rate × data missing cost
   = 50 + 30% x 50 = 65
   $$CPI_{None} = CPI_{benchmark} + Pause\ Cycles = 1 + 65 = 66$$
   2) Only L1-level cache
   Pause Cycles = instruction missing rate × instruction missing cost + data missing rate × data missing cost
   = 2% x 50 + 30% x 2% x 50
   = 1 + 0.3
   = 1.3
   $$CPI_{L1} = CPI_{benchmark} + Pause\ Cycles = 1 + 1.3 = 2.3$$
   3) Only L2-level cache
   Because the instruction cache and data cache are not distinguished, the cost of L1 instruction missing is the same as the cost of L1 data missing.
   Pause Cycles = instruction missing rate × instruction missing cost + data missing rate × data missing cost
   = 2% × L1 missing cost + 30% × 2% × L1 missing cost
   = (2% + 0.6%) × (10 + 5% × 50)
   = 0.026 × 12.5
   = 0.325
   $$CPI_{L2} = CPI_{benchmark} + Pause\ Cycles = 1 + 0.325 = 1.325$$

2. The direct mapping cache parameters are as follows: the data capacity of the cache is 16KB, and the cache block is 16B. The cache is now connected to a 32-bit CPU. Please provide the TIO structure of the cache and the cache block size after the tag is included.

   Final Answer:

   1) Offset: log2(16) = 4

2) Index: 16KB/16 = 1024 Blocks, log2(1024) = 10
3) Tag: 32 – 4 – 10 = 18
4) Total cache block capacity: data + Tag + Valid = 16B + 18b + 1b = 16B + 19b

3. Group associative mapping cache parameters are as follows: the data capacity of the cache is 512KB, 16-way associative, and the cache block is 32B. The cache is now connected to a 32-bit CPU. Please give the TIO structure of the cache.

   Final Answer:
   1) Offset: log2(32) = 5
   2) Index: 512KB/32B = 16K Blocks, 16K/16 = 1K Groups, log2(1K) = 10
   3) Tag: 32 – 5 – 10 = 17

4. For the third system, the cache uses a write-back strategy. The cache hit time is 1 clock cycle. The cost of transferring a block from main memory or writing back a block is 100 clock cycles. A program snippet initializes all 1MB memory cells with a start address of 0000_0000h to 0. The code is shown below.

```
int *p=0x0 ;
for ( int i=0; i<1024*1024/4; i++ )
        *p++ = 0 ;
```

   1) Calculate the missing data access rate in this program fragment (ignoring factors such as missing instructions).
   2) Calculate how many clock cycles are actually needed for data access of this program fragment.
   3) Calculate the ratio of the number of program memory accesses to the actual number of clock cycles.

   Final Answer:
   1) i. Number of blocks corresponding to 1MB main memory: 1MB / 32B = 32K (block)
      ii. Because it is linearly initialized, these 32K blocks are accessed once. Can be seen as the following loop:
      iii. First cycle:
         1. Missing when initializing main memory block 0 word 0. So from the main memory block 0 to the cache group 0 block
      0. The next 7 words (8 words-1 words) all hit.
         2. Missing when initializing main memory block 1 word 0. So from the group 1 block that writes the main memory block 1 to the cache
      0. The next 7 words (8 words-1 words) all hit.
         3. Similarly, main memory block 15 is mapped to cache group 15 block 0.
      iv. Second cycle:
         1. When the main memory block is 16 words 0, it is missing and will be mapped to cache group 0 block 1; after that, all 7 words will hit.
         2. By analogy, when the main memory block 31 words 0 is missing, it will be mapped to cache group 15 block 1. The next 7 words hit.

v. Repeat the process until the 1Kth cycle. In the 1Kth cycle, the main memory block (16K-16) to the main memory block (16K-1) are loaded into cache group 0 block 15 and cache group 15 block 15 respectively.
vi. At this point, the first 16K blocks of main memory have been accessed, all cache blocks have been filled, and there are no free blocks.
vii. The behavior of the last 16K blocks of main memory is exactly the same as that of the first 16K blocks, except that the first 16K blocks need to be written back to the main memory.
viii. When the main memory is accessed, the access situation of each corresponding main memory block is exactly the same: word 0 is missing, and words 1 to 7 are hit. Therefore, the missing rate of data access = 1/8 = 12.5%

2) i. For each block, there is 1 miss, at the cost of 100 clock cycles; 7 hits, 7 cycles. So each block is 100 + 7 = 107 clock cycles.
ii. There are a total of 32K blocks, so the access time = 107 * 32K (clock cycle)
iii. But the last 16K blocks will also cause 16K replacements, each replacement will have 100 clock cycles, so the replacement time = 100 * 16K (clock cycles)
iv. Total time = 107 * 32K + 100 * 16K = 5024K (clock cycle)
3) i. Total program memory accesses = 1MB / 4 = 256K
ii. Total program memory access times: actual clock cycles = 256K: 5024K ≈ 1:20
iii. This means that memory accesses can only be completed every 20 clock cycles.

5. Assume that the virtual and physical page sizes of a system are 8KB, 40-bit virtual addresses, and the physical main memory capacity is 32GB. What are the digits of the virtual page number and the physical page number?

Final Answer:
1) 8KB pages, the page address is 13 bits.
2) Number of virtual page numbers (VPN) = 40-13 = 27 digits
3) Number of physical page numbers (PPN) = log2 (32G) -13 = 35-13 = 22 digits

6. The designer designed TLB for the system in question 5. TLB uses a 2-way group associative structure with a total of 256 pages table entries. The structure of each page table entry in TLB is shown in the figure below.

Valid	Dirty	Access Permission	TLB Tag	PPN
1 bit	1 bit	2 bits		

1) What is the number of digits of each page table entry?
2) What is the total storage capacity of the page table?
3) The operating system design team hopes to reduce the page size from 8KB to 4KB, but the hardware design team believes that it will increase hardware overhead and therefore does not agree to change the page size. As the head of the hardware designer team, please state your reasons.

Final Answer:
1) Number of TLB groups = 256/2 = 128 groups
2) Index number of TLB = log2 (128) = 7
3) TLB tag bits = virtual address bits-page offset number-index bits = 40-13-7 = 20 bits

4) Number of page table entries = 1 + 1 + 2 + 20 + 22 = 46 bits

Valid	Dirty	Access Permission	TLB Tag	PPN
1 bit	1 bit	2 bits	20	22

7. There are 4K × 8-bit DRAM memory chips.
   1) How many row / column selection lines are output by the built-in decoder of the DRAM chip.
   2) How many bits of the refresh address counter are built into the DRAM chip.
   3) How many bits are refreshed in a row of memory cell in each refresh cycle of DRAM.
   4) The main memory capacity is 8K × 8 bits, which requires the number of DRAM chips.
   5) Give the logic expression of the chip select control signal of each DRAM chip.

   Final Answer:

   1) The DRAM chip has a total of 4K cells, so the number of bits of the internal address line = log2 (4K) = 12 means that the internal address is A [11: 0]. Because DRAM uses row address and column address sharing, the row / column address selection lines are both 12/2 = 6.
   2) Because DRAM is refreshed one row at a time, the number of bits in the refresh address counter is the same as the number of bits in the row address, that is, the number of bits in the refresh address counter is 6 bits.
   3) The number of cells in a row is determined by the number of column addresses, that is, the number of cells in a row = 26 = 64. Therefore, the total number of memory cell bits is 64x8 = 512 bits.
   4) The required number of DRAM chips = (8K x 8) / (4K x 8) = 2 pieces.
   5) The main memory address is A [12: 0]. The number of bits per DRAM chip is A [11: 0], so A12 is used to select 2 chips. Assuming chip 1 corresponds to 0 ~ 4K and chip 2 corresponds to 4K-8K, then:
      Chip Select for Chip 0 =! A12
      Chip Select for Chip 1 = A12