计算机组成与设计
习题

信电学院信通所 孙佳科
22131087@zju.edu.cn
18888923044
浙江大学玉泉校区信电楼308室

HW1

Problem 1

Add comments to the following code and describe in one sentence what it computes. Assume that a0 is used for the input and initially contains n, a positive integer. Assume that a0 is used for the output.

Code for Problem 1:

```
begin:    addi t0, x0, 0          # t0 = 0
          addi t1, x0, 1          # t1 = 1
loop:     slt t2, a0, t1          # t2 = a0 < t1 ? 1 : 0
          bne t2, x0, finish      # jump to finish, if  t2 != 0
          add t0, t0, t1          # t0 = t0 + t1
          addi t1, t1, 2          # t1 = t1 + 2
          j loop                  # jump to loop
finish:   add a0, t0, x0          # a0 = t0
```

计算所有不大于 n 的奇数之和

HW1

Problem 2

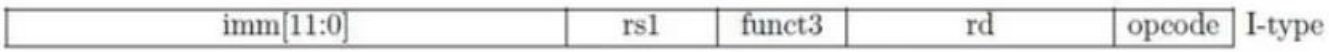
Assume we have an array in memory that contains int* arr = {1,2,3,4,5,6,0}. Let the values of arr be a multiple of 4 and stored in register s0. What do the snippets of RISC-V code do? Add comments to the following code.
Assume that all the instructions are run one after the other in the same context.

a)
```
lw t0, 12(s0)          # t0 = arr[3] = 4
```

b)
```
slli t1, t0, 2         # t1 = t0<<2 = 16
add t2, s0, t1         # t2 = s0 + t1
lw t3, 0(t2)           # t3 = arr[4] = 5
addi t3, t3, 1         # t3 = t3 + 1 = 6
sw t3, 0(t2)           # arr[4] = $t3 = 6
```

c)
```
lw t0, 0(s0)           # t0 = arr[0]
xori t0, t0, 0xFFF     # t0 = t0 ^ 0xFFF = ~t0
addi t0, t0, 1         # t0 = t0 + 1
```

I-type指令立即数是12位，符号扩展到32。



HW2

Problem 1

Write a loop that reverses order of the bits of an 8-bit number in s0 and stores the result in s1. RISC-V registers have 4 bytes, but in this problem, the higher 24 bits of s0 and s1 are always 0. For example, if the lower 8 bits of s0 are 00001101, the lower 8 bites s1 should be 10110000.

```
pseudocode:
s1 = 0;                     //结果初始化为 0
for(t0=8; t0>0; t0--){
    t1 = s0 & 0x00000001    // t1 = s0 的最低位
    s0 = s0>>1              //s0 右移 1bit
    s1 = (s1<<1) + t1       //将 t1 添加到 s1 的最低位
}
```

```
begin:  addi t0, x0, 8          # t0 = 8
        addi s1, x0, 0          # s1 = 0
loop:   beq t0, x0, finish      # jump to finish, if t0 = 0
        andi t1, s0, 1          # t1 = s0 & 0x00000001
        srli s0, s0, 1          # s0 = s0 >> 1
        slli s1, s1, 1          # s1 = s1 << 1
        add s1, s1, t1          # s1 = s1 + t1
        addi t0, t0, -1         # t0 = t0 - 1
        jal x0, loop            # jump to loop
finish:
```


4

HW2

Problem 2

The following functions are syntactically-correct C, but written in an incomprehensible style. Describe the behavior of each function.

c) Recall that ^ is the bitwise exclusive-or (XOR) operator.

```
1  void baz(int x, int y) {
2              x = x ^ y;
3              y = x ^ y;
4              x = x ^ y;
5    }
```

Ultimately does not change the value of either x or y.

形参：在函数定义中出现的参数，只有在函数被调用时才会分配内存，调用结束后，立刻释放内存，形参变量只有在函数内部有效，不能在函数外部使用。

实参：函数被调用时给出的参数包含了实实在在的数据，会被函数内部的代码使用。

形参和实参可以同名，但它们之间是相互独立的，互不影响，因为实参在函数外部有效，而形参在函数内部有效

HW2

```
1  void baz(int *x, int *y) {
2              int temp = *x;
3              *x  =  *x  ^  *y;
3              *y =  *x  ^ * y;
4              *x =  *x  ^ * y;
5    }
```

形参用指针形式

x, y 指向地址

*x，*y是x, y地址中的值，

改变地址中的值可以改变实参

HW2

Problem 3

For each part, choose one or more of the following memory segments where the data could be located: code, static, heap, stack.

(a) Static variables
static
(b) Local variables
stack
(c) Global variables
static
(d) Constants
code, static, stack
(e) Machine Instructions
code
(f) Result of malloc
heap
(g) String Literals
static, stack

栈区：由编译器自动分配释放，存放函数的参数值、返回值和局部变量。

堆区：堆是由程序员自行分配的内存块，堆的申请释放工作由程序员控制

静态内存分配：
BBS区：存放的是未初始化的全局变量和静态变量。
数据区：存放已初始化的全局变量、静态变量（全局和局部）、常量数据。

代码区：存放CPU执行的机器指令，代码区是可共享，并且是只读的。

HW3

Problem 1

Given the following RISC -V assembly code (and assuming all registers start at 0):

```
            addi x1, x0, 5        # x1 = 5
            add x2, x1, x1        # x2 = x1 + x1  = 10
repeat:     addi x2, x2, -3       # x2 = x2 - 3
            add x3, x2, x2        # x3 = x2 + x2
            addi x1, x1, -1       # x1 = x1 - 1
            bne x0, x1, repeat    # jump to repeat if x1 != 0
```

1.How many times is the line, “ add x3, x2, x2 ” executed?
2.What is the final value of x3?

1. 5 times
2. -10

8

HW3

Problem 2

Translate the following binary code into the corresponding risc-v instruction. **(Refer to page 119 of the textbook)**

1. 0000111 00001 00010 010 00000 0100011 sw x1, 224(x2)
2. 0000111 00000 00010 010 00001 0000011 lw x1, 224(x2)
3. 0000111 00000 00010 000 00001 0010011 addi x1, x2, 224
4. 0100000 00011 00010 000 00001 0110011 sub x1, x2, x3
5. 0000000 00011 00010 000 00001 0110011 add x1, x2, x3

9

HW4

Problem 1

There are 3 different processors P1, P2 and P3 with the same ISA, its clock frequencies and CPIs are as follow:

Processor	Clock Frequency	CPI
P1	2 GHz	1.5
P2	1.5 GHz	1.2
P3	3 GHz	2.0

1. Calculate the MIPS for each processor and find out which one has the best performance?
2. If we want to increase the MIPS by 30%, which will cause an increase of CPI by 15%, what's the clock frequency should be for each processor?

1. $MIPS = \frac{\text{Clock Frequency}}{CPI} \times 10^{-6}$

P1: 1333 MIPS
P2: 1250 MIPS
P3: 1500 MIPS
So, P3 has the best performance.

2. Above formula can be transformed to $\text{Clock Frequency} = MIPS \times CPI \times 10^{6}$

P1: $\text{Clock Frequency} = 1333 \times 1.3 \times 1.5 \times 1.15 \times 10^{6} \approx 2.99GHz$

P2: $\text{Clock Frequency} = 1250 \times 1.3 \times 1.2 \times 1.15 \times 10^{6} \approx 2.24GHz$

P3: $\text{Clock Frequency} = 1500 \times 1.3 \times 2 \times 1.15 \times 10^{6} \approx 4.49GHz$

HW4

Problem 2

The table below shows the instruction type breakdown of a given application executed on 1, 2, 4, or 8 processors. Using this data, you will be exploring the speedup of applications on parallel processors.

Processors	No. Instructions per processor			CPI		
	Arithmetic	Load/Store	Branch	Arithmetic	Load/Store	Branch
1	2560	1280	256	1	4	2
2	1280	640	128	1	5	2
4	640	320	64	1	7	2
8	320	160	32	1	12	2

1. The table above shows the number of instructions required per processor to complete a program on a multiprocessor with 1, 2, 4, or 8 processors. What is the total number of instructions executed per processor? What is the aggregate number of instructions executed across all processors?

1.1 . a)

Processors	Instructions per processor	Total instructions
1	2560+1280+256 = 4096	4096
2	1280+640+128 = 2048	2048*2=4096
4	1024	4096
8	512	4096

HW4

Problem 2

1.2 Given the CPI values on the right of the table above, find the total execution time for this program on 1, 2, 4, and 8 processors. Assume that each processor has a 2 GHz clock frequency.

1.2 .a)

Processors	Execution Time
1	(2560*1+1280*4+256*2)/2G = 4.096*10^-6 s
2	2.368*10^-6 s
4	1.504*10^-6 s
8	1.152*10^-6 s

1.3 If the CPI of the arithmetic instructions was doubled, what would the impact be on the execution time of the program on 1, 2, 4, or 8 processors? Assume that each processor has a 2 GHz clock frequency.

1.3 . a)

Processors	Execution Time
1	(2560*2+1280*4+256*2)/2G = 5.376*10^-6 s
2	(1280*2+640*5+128*2)/2G = 3.008*10^-6 s
4	1.824*10^-6 s
8	1.312*10^-6 s

HW4

Problem 2

The table below shows the number of instructions per processor core on a multicore processor as well as the average CPI for executing the program on 1, 2, 4, or 8 cores. Using this data, you will be exploring the speedup of applications on multicore processors.

Cores per Processor	Instructions per Core	Average CPI
1	1.00E+**10**	1.2
2	5.00E+**9**	1.4
4	2.50E+**9**	1.8
8	1.25E+**9**	2.6

1.4 Assuming a 3 GHz clock frequency, what is the execution time of the program using 1, 2, 4, or 8 cores?

1.4 .a)

Processors	Execution Time
1	(1.00*10^10*1.2)/3G = 4 s
2	(5.00*10^9*1.4)/3G = 2.33 s
4	1.5 s
8	1.08 s

HW4

Problem 2

1.6 If using a single core, find the required CPI for this core to get an execution time equal to the time obtained by using the number of cores in the table above (execution times in problem 1.4). Note that the number of instructions should be the aggregate number of instructions executed across all the cores.

1.5 .a)

Processors	Required CPI
1	1.2
2	1.4/2= 0.7
4	1.8/4=0.45
8	2.6/8=0.325

HW5

Problem 1

clock cycle time of the entire datapath, and how these components are utilized by instructions. For problems in thisexercise, assume the following latencies for logic blocks in the datapath:

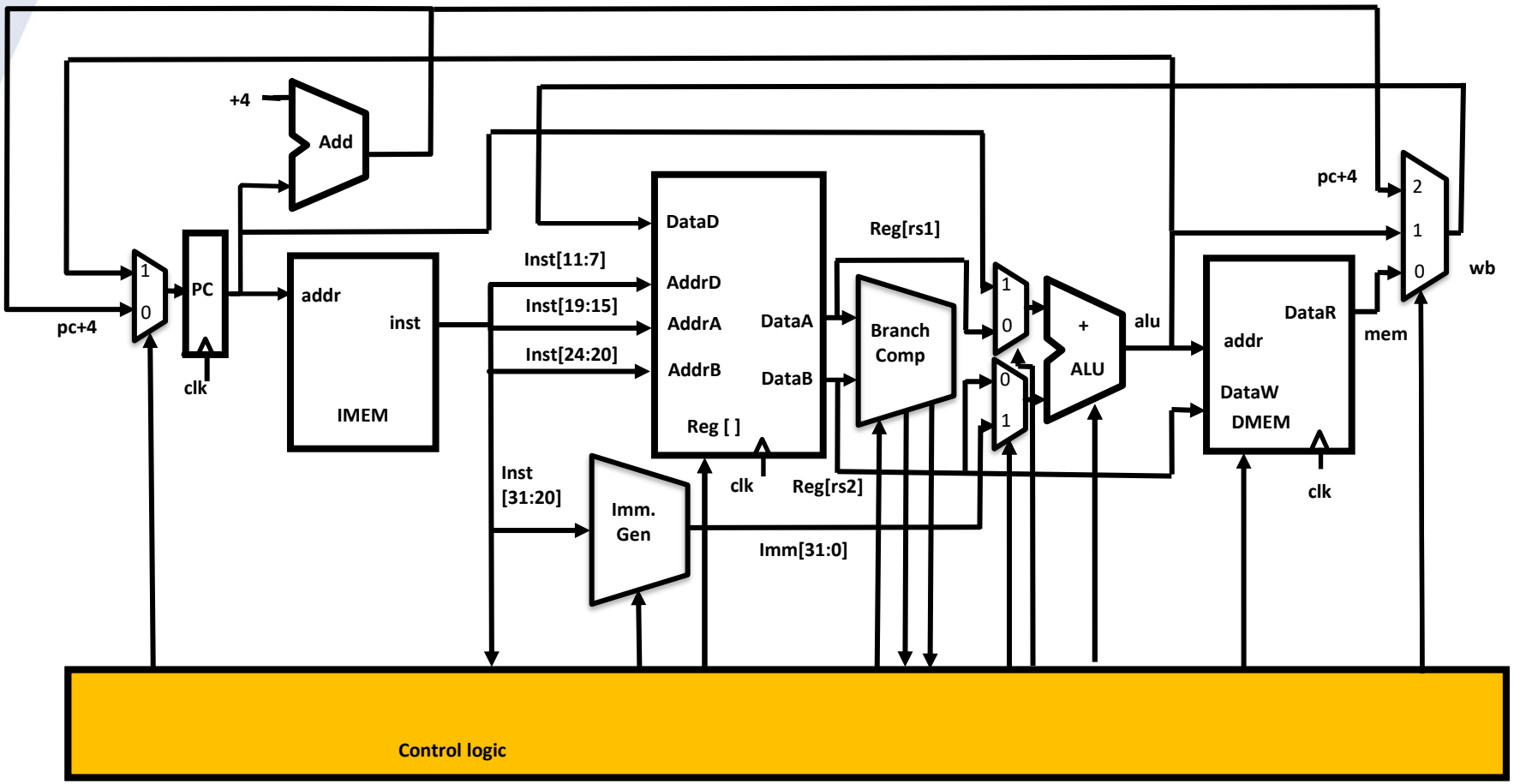
I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-extend	Shift-left-2
400ps	100ps	30ps	120ps	200ps	350ps	20ps	0ps

a) What is the clock cycle time if the only types of instructions we need to support are ALU instructions (ADD, AND, etc.)?

b) What is the clock cycle time if we only have to support LW instructions?

c) What is the clock cycle time if we must support ADD, BEQ, LW, and SW instructions?

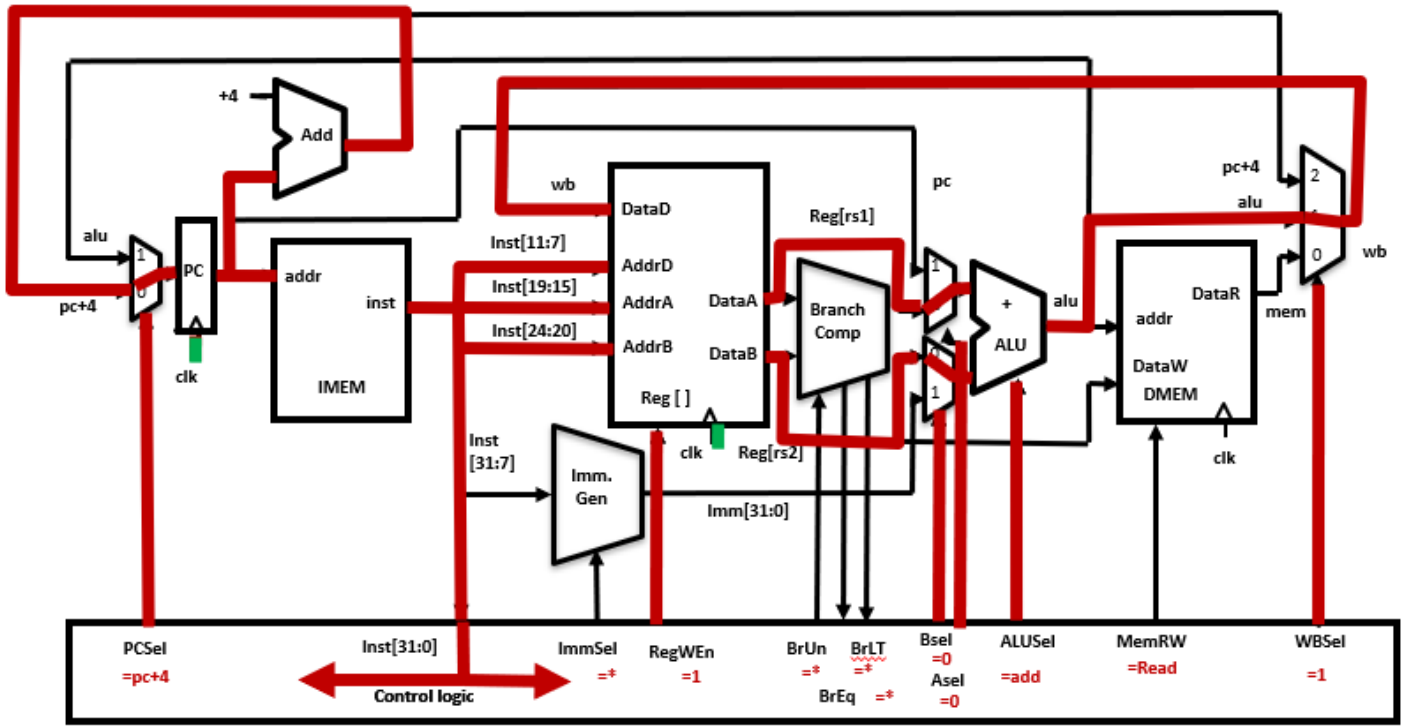
HW5


+4
Add
pc+4
1
0
PC
clk
addr
inst
IMEM
Inst[11:7]
Inst[19:15]
Inst[24:20]
DataD
AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]
clk
Reg[rs1]
Reg[rs2]
Branch Comp
Inst [31:20]
Imm. Gen
Imm[31:0]
1
0
0
1
+
ALU
alu
pc+4
2
1
0
wb
DataR
addr
DataW
DMEM
clk
mem
Control logic

HW5

解：对于一条指令而言，关键路径（产生最长延迟的那条路径）上各单元的延迟决定了该指令的最小延迟

ALU指令：



I-Mem->Regs->ALUmux->ALU->Mux->Regs

HW5

(a) 对于ALU指令（ADD, AND等）需要：

I-Mem->Regs->ALUmux->ALU->Mux->Regs

所以时钟周期：

400 (I-Mem) + 200 (Reg. Read) + 30 (Mux) + 120 (ALU) + 30 (Mux) + 200 (Reg. Write) = 980 ps

这里，add（pc+4）不算在关键路径中。

HW5

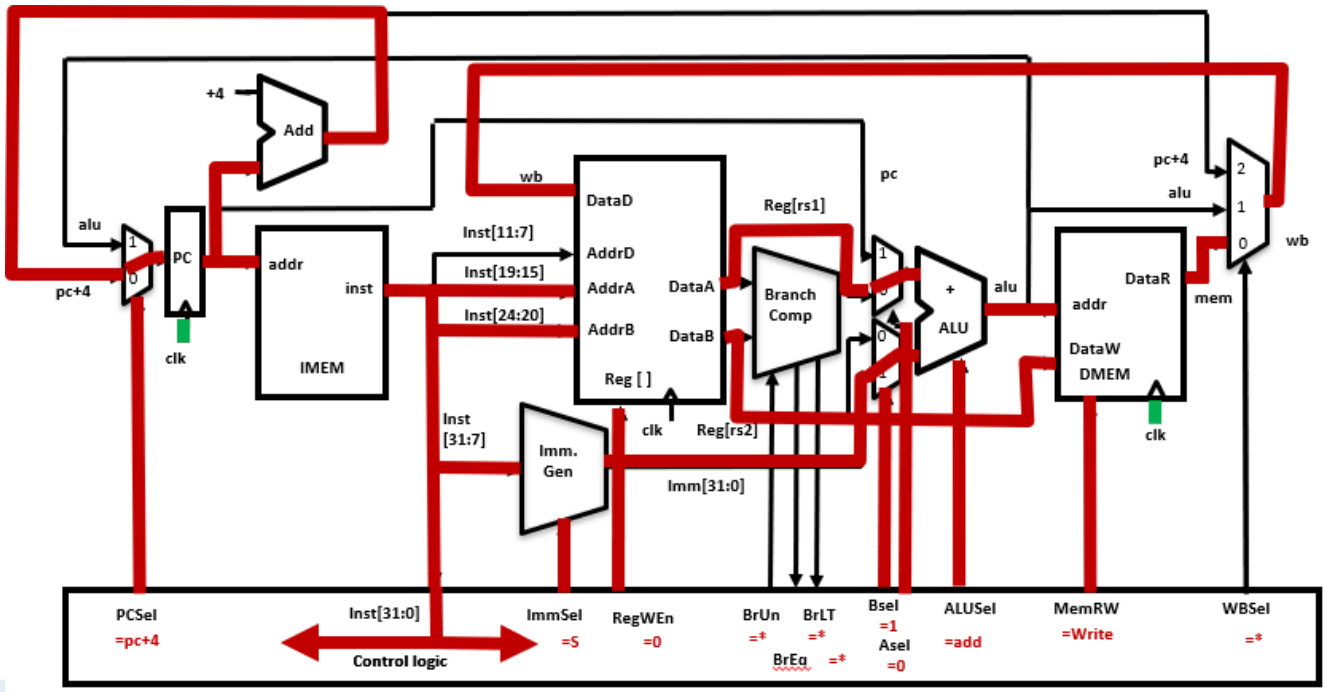
(b) 对于装载指令(LW)，其需要符号扩展和数据存储器访问，其长路径为：

I-Mem -> Regs -> ALUmux -> ALU -> D-mem -> Mux -> Regs

符号扩展延迟小于寄存器延迟

所以关键路径时延：

400ps(I-Mem) + 200ps(Reg.read) + 30ps(ALUmux) + 120ps(ALU) + 350ps(D-mem) + 30ps(Mux) + 200ps(Reg.write) = 1330ps



HW5

- C) 要求支持 ADD, BEQ, LW, and SW 指令，这里我们需要“最慢”的指令(LW)
- 1330ps

HW5

For the remaining problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

ADD	ADDI	NOT	BEQ	LW	SW
30%	15%	5%	20%	20%	10%

d) In what fraction of all cycles is the data memory used?

e) In what fraction of all cycles is the input of the sign-extend circuit needed?

f) If we can improve the latency of one of the given datapath components by 10%, which component should it be? What is the speedup from this improvement?

- **d) 需要访问D-Mem的指令：LW、SW**
  - 20% + 10% = 30%
- **e) ADDI ：对立即数做符号位扩展**

  BEQ、LW、SW：对偏移地址做符号位扩展
  - 15% + 20% + 20% + 10% = 65%
- **f) 时钟由关键路径时延（LW）决定，**

  关键路径：I-Mem->Regs->Mux->ALU->D-Mem->Mux->Regs

I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-extend	Shift-left-2
400ps	100ps	30ps	120ps	200ps	350ps	20ps	0ps

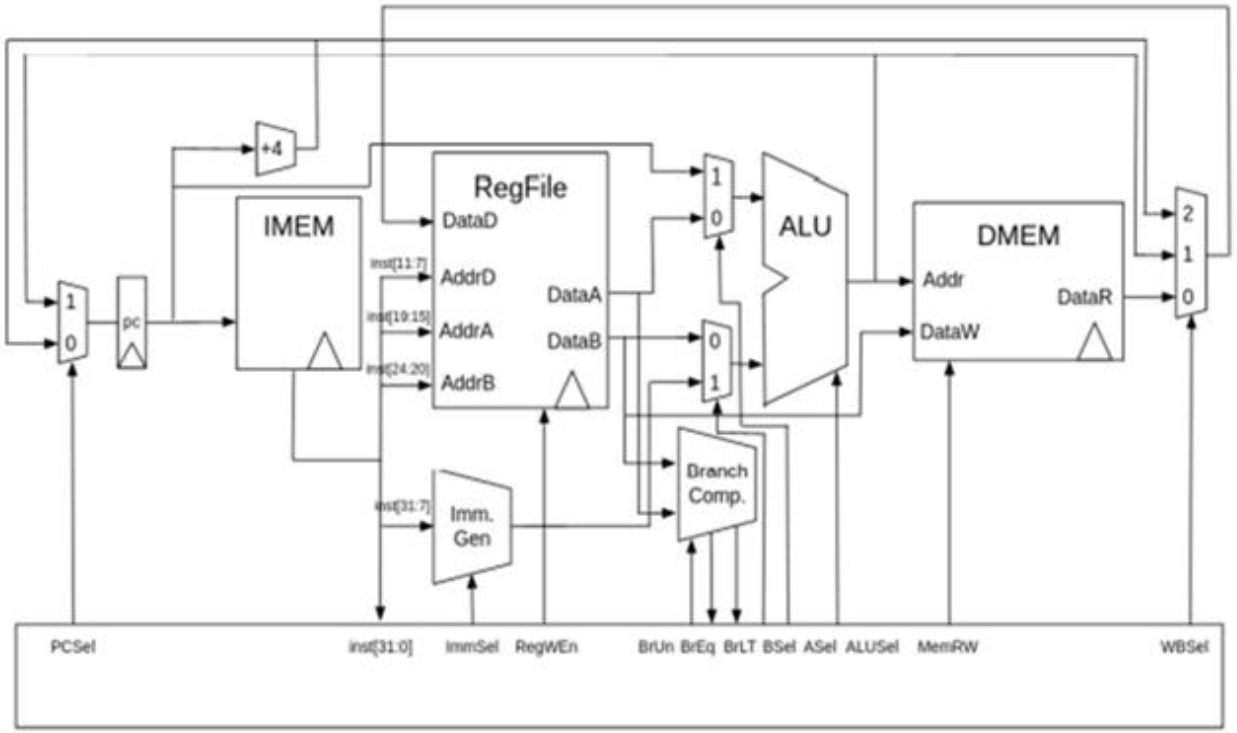
400ps*10%=40ps

Speed-up : 1330 / (1330-40) = 1.031

HW5

Problem 2

Given the standard single-cycle RISC-V datapath (Figure 1) for the following instruction, what control signals are provided for the datapath? Use 0, 1, or * for signals and N/A if the given signal is not provided by the control logic; for the ALU, assume 0 refers to the add operation and 1 to the sub operation. For the immediate selector, use I for immediate-type, S for store-type, B for branch-type, J for jump-type, and U for upper-immediate type. **Use the signals without any leading or trailing whitespace otherwise it may be marked incorrect. All answers should be exactly one signal.**



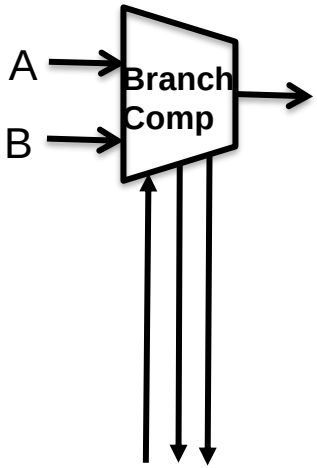
HW5

Problem 2

```
PC = PC + 4, branch not taken          (PCSel = 0)
     PC + immediate, branch taken      (PCSel = 1)
```

compute `PC + immediate`

compare values of `rs1` and `rs2`



- **BrEq = 1, if A=B**
- **BrLT = 1, if A < B**
- **BrUn = 0 signed comparison**
- **BrUn =1 selects unsigned comparison (bltu, bgeu)**

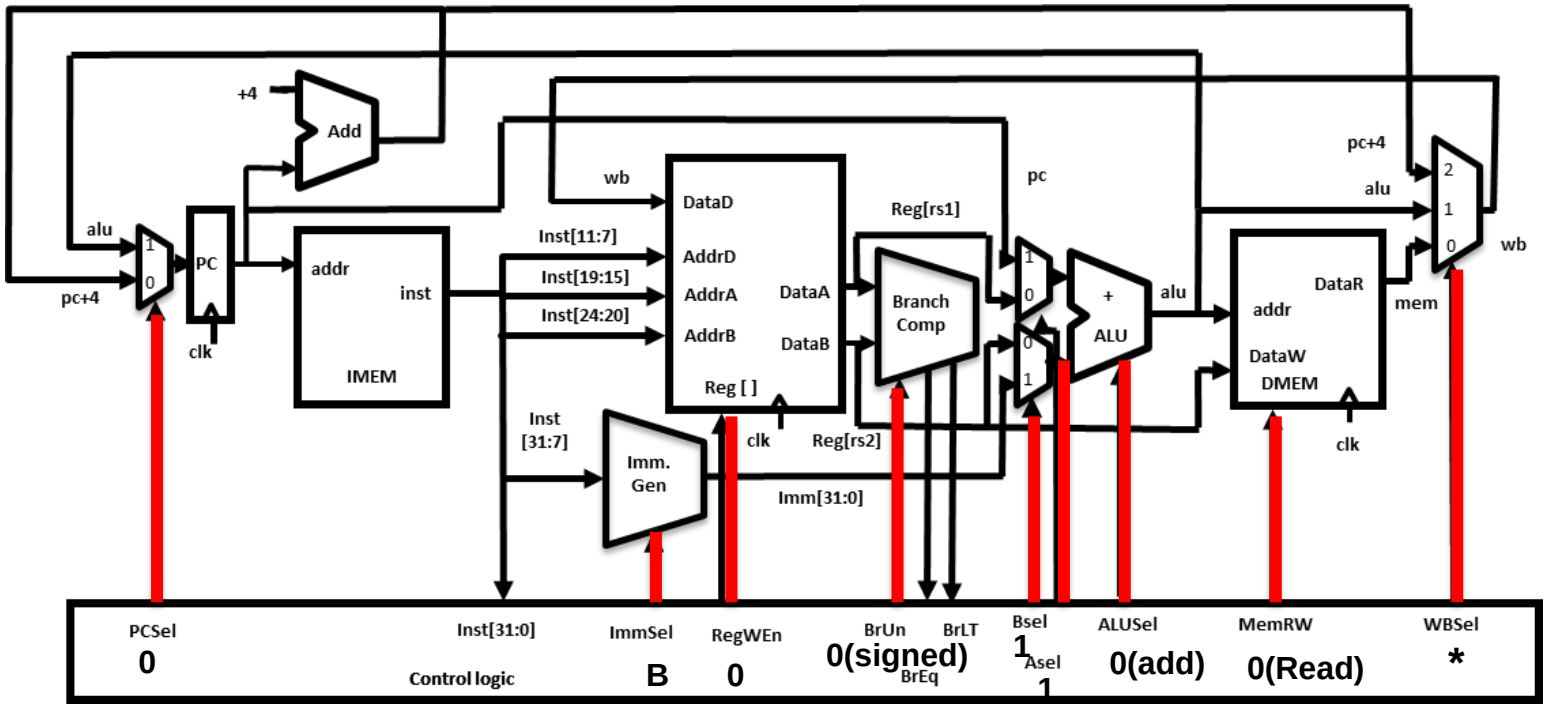
HW5

Instruction : blt r1, r2, Label

1) Not taken PCSel = 0

2) Taken PCSel = 1

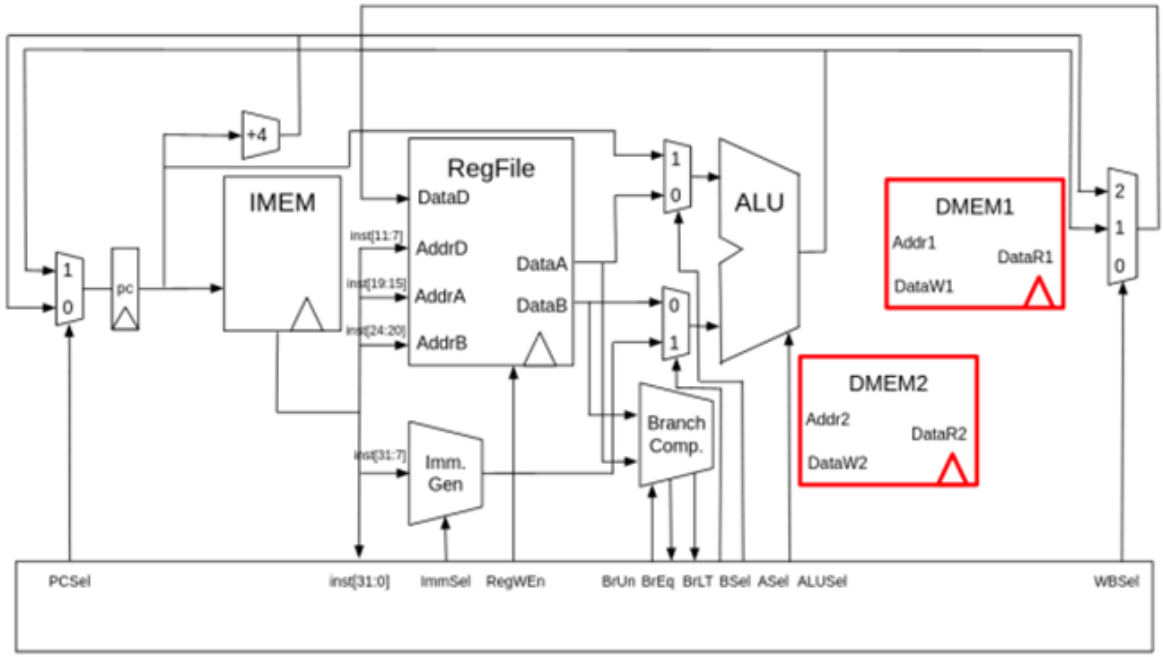
Inst[31:0], BrEq and BrLT given from datapath to control logic are N/A



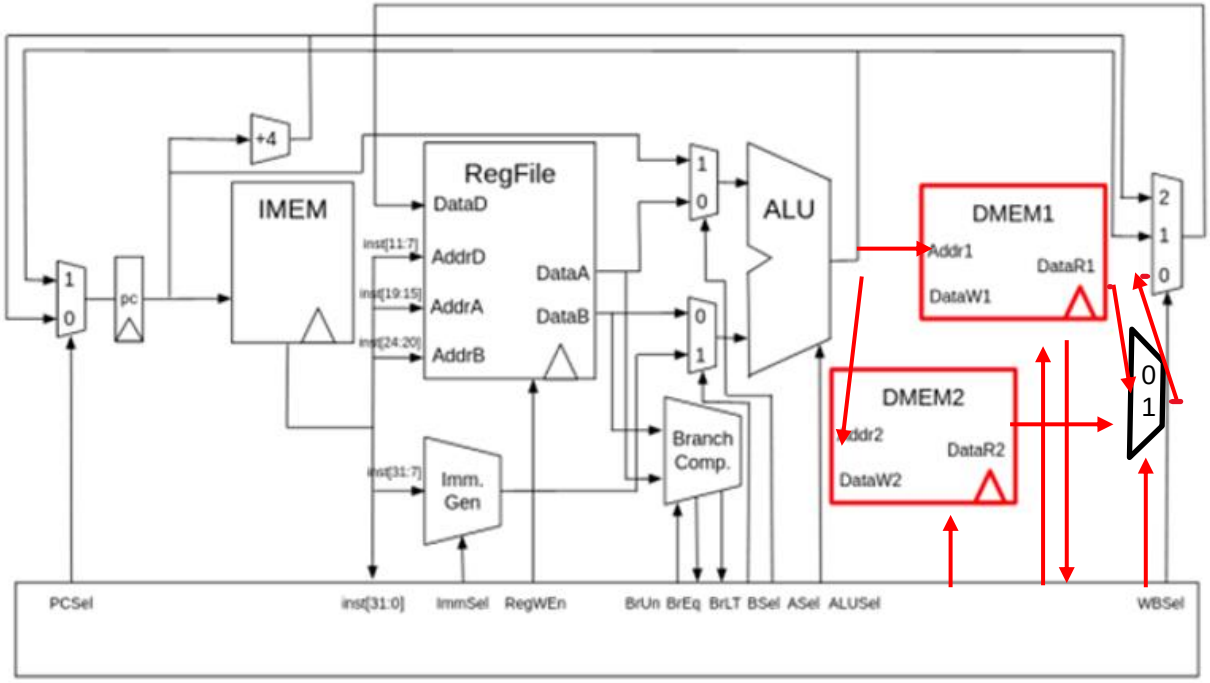
HW5

Problem 4

Say we alter our datapath such that we split data memory into two serial steps, i.e. instead of MEM we have MEM1 and MEM2 (see Figure 2) which are respectively the first and second half of the original memory. The way this new datapath operates is that any memory access first must go through MEM1; if it finds the data it needs there, then the datapath bypasses the second memory access and does not attempt to make an access; the value obtained from MEM1 is then passed to the writeback mux through the MEM slot. However, if the address is not in MEM1, then the datapath signals to control logic that the ALU needs to pass the address to MEM2 in order to make an access there, blocking any subsequent ALU operations for that instruction, and the result is fed into the MEM input in the writeback mux. Accessing MEM1 and MEM2 individually takes the same amount of delay.



HW5



- ☐ Path from MEM1 output to MEM2 input.
- ☑ Path from ALU to MEM1 input.
- ☐ Add additional state element.
- ☑ Path from control logic to additional mux.
- ☑ Path from ALU to MEM2 input.
- ☑ Path from control logic to MEM1.
- ☑ Path from MEM2 out to WB mux.
- ☑ Path from control logic to MEM2.
- ☐ Path from MEM2 to control logic.
- ☑ Path from MEM1 to control logic.
- ☑ Add additional mux.
- ☐ Path from control logic to additional state element.
- ☑ Path from MEM1 to WB mux.

2, 4, 5, 6, 7, 8, 10, 11, 13

HW6

Problem 1

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

IF	ID	EX	MEM	WB
250ps	400ps	150ps	350ps	200ps

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

alu	beq	lw	sw
40%	20%	25%	15%

a) What is the clock cycle time in a pipelined and non-pipelined processor?
b) What is the total latency of an LW instruction in a pipelined and non-pipelined processor?
c) If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

在流水线结构中，时钟周期是最长阶段的时延。

在 5 级流水线中，一条指令需要五个时钟周期才能运行结束，所以此时一条指令完成的时延是时钟周期的五倍。

在单周期的设计中，一个时钟周期执行完成一条指令，故一条指令的时延是一个时钟周期。

- **a) pipelined: 400ps single-cycle : 250ps + 400ps + 150ps + 350ps + 200ps = 1350ps**
- **b) pipelined: 400ps*5=2000ps single-cycle : 1350ps**
- **c) ID 350ps**

HW6

Problem 2

Look for data hazards in the code below, and figure out how forwarding could be used to solve them. them.

Instruction	C1	C2	C3	C4	C5	C6	C7
1. add t0, a0, -1	IF	ID	EX	MEM	WB		
2. and s2, t0, a0		IF	ID	EX	MEM	WB	
3. sltiu a0, t0, 5			IF	ID	EX	MEM	WB

Answer:

There are two data hazards, between instructions 1 and 2, and between instructions 1 and 3. The first could be resolved by forwarding the result of the EX stage in C3 to the beginning of the EX stage in C4, and the second could be resolved by forwarding the result of the EX stage in C3 to the beginning of the EX stage in C5.

HW6

Problem 3

This exercise is intended to help you understand the relationship between forwarding, hazard detection, and ISA design. Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a 5-stage pipelined datapath:

```
add x5, x2, x1
lw x3, 4(x5)
or x3, x5, x3
lw x2, 0(x2)
sw x3, 0(x5)
```

1) if there is no forwarding or hazard detection, insert nop to ensure correct execution.
2) Repeat 1) but now use nop only when a hazard cannot be avoided by changing or rearranging these instructions. You can assume register x7 can be used to hold temporary values in your modified cade.
3) If the processor has forwarding, but we forget to implement the hazard detection unit, what happens when this code executes?

HW6

Problem 3

1) if there is no forwarding or hazard detection, insert nop to ensure correct execution.

1）

```
add x5, x2, x1
lw x3, 4(x5)
or x3, x5, x3
lw x2, 0(x2)
sw x3, 0(x5)
```

Instruction	C1	C2	C3	C4	C5	C6
1. add x5, x2, x1	IF	ID	EX	MEM	WB	
2. lw x3, 4(x5)		IF	ID	EX	MEM	WB

```
ADD x5, x2, x1
NOP
NOP
LW x3, 4(x5)
NOP
NOP
OR x3, x5, x3
LW x2, 0(x2)
NOP
SW X3, 0(x5)
```

Instruction	C1	C2	C3	C4	C5	C6
1. add x5, x2, x1	IF	ID	EX	MEM	WB	
2. NOP						
3. NOP						
4. lw x3, 4(x5)				IF	ID	MEM

HW6

2) Repeat 1) but now use nop only when a hazard cannot be avoided by changing or rearranging these instructions. You can assume register x7 can be used to hold temporary values in your modified cade.

2）

```
add x5, x2, x1
lw x3, 4(x5)
or x3, x5, x3
lw x2, 0(x2)
sw x3, 0(x5)
```

```
ADD x5, x2, x1
LW x2, 0(x2)
NOP
LW x3, 4(x5)
NOP
NOP
OR x3, x5, x3
NOP
NOP
SW X3, 0(x5)
```

HW6

3) If the processor has forwarding, but we forget to implement the hazard detection unit, what happens when this code executes?

3）With forwarding, the hazard detection unit is still needed because it must insert a one-cycle stall whenever the load supplies a value to the instruction that immediately follows that load. Without the hazard detection unit, the instruction that depends on the immediately preceding load gets the stale value the register had before the load instruction.

Instruction	C1	C2	C3	C4	C5	C6
lw x3, 4(x5)	IF	ID	EX	MEM	WB	
or x3, x5,x3		IF	ID	EX	MEM	WB

lw x3, 4(x5)
or x3, x5, x3

Instruction	C1	C2	C3	C4	C5	C6	C7
lw x3, 4(x5)	IF	ID	EX	MEM	WB		
NOP		IF	ID	EX	MEM	WB	
or x3, x5, x3			IF	ID	EX	MEM	WB

The register x3 in instruction “or x3, x5, x3” has the stale value instead of the value the load instruction (LW x3, 4(x5)) supplies.

HW6

Problem 4

In the conventional fully bypassed 5-stage pipeline discussed in class, the main remaining data hazard is the load-use hazard. For example, in the following instruction sequence:

```
lw x1, 16(x2)
xor x3, x1, x5
```

The xor instruction will experience a one-cycle stall in the decode stage, while the lw propagates to the memory stage. For the remainder of this question, we will only consider 32-bit loads (lw). ***You may ignore branch and jump instructions in this problem.***

```
F   D   X   M   W                   lw
    F   D   D   X   M   W           xor
```

One approach to removing the hazard is to speculate on the load value returned (*load-value speculation*). In lecture, we briefly described one scheme, a *load-zero predictor* where the value was predicted to be zero. An instruction that would otherwise stall in the decode stage waiting for a memory value was provided the value 0 instead. When the load instruction reaches the memory stage, the value is checked and if not zero, the pipeline is flushed and the dependent instruction is replayed as shown.

```
F   D   X   M   W                       lw
    F   D   X   -   -                   xor flushed
        F   D   -   -   -               flushed
            F   -   -   -   -           flushed
                F   D   X   M   W       xor correct value
```

HW6

Problem 4

Q3.A [3 points] Assuming the pipeline is flushed as shown on a mispredict, calculate the minimum accuracy required for the load-zero predictor to improve performance. Note the predictor is only used when an instruction would otherwise stall in decode waiting for a load value.

Without load-zero predictor
Needs one cycle stall on the decode stage.
Totally 7 cycles.

F	D	X	M	W			lw
	F	D	D	X	M	W	xor

设α为最小预测准确率

$$6\alpha + 9(1-\alpha) = 7$$

$$\alpha = \frac{2}{3} \approx 66.7\ \%$$

With load-zero predictor
(predict correctly) 6 cycles.
(predict incorrectly) 9 cycles.

F	D	X	M	W					lw
	F	D	X	-	-				xor flushed
		F	D	-	-	-			flushed
			F	-	-	-	-		flushed
				F	D	X	M	W	xor correct value

Mispredict penalty of a predicted load-zero is 3 cycles. The load-use delay is 1 cycle. The predictor should be at least 66.7% accurate to improve performance.

HW7

Problem1

Computes the floating point number represented by the binary below, and write the decimal expression.

$$(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$$

2.

S	Exponent	Significand
1	1111 1111	0000 0000 0000 0000 0000 000

$\Longrightarrow$ $-\infty$

3.

S	Exponent	Significand
1	1100 0000	100 0000 0000 0000 0000 000

$$(-1)^1 \times (1 + .1)_2 \times 2^{(192-127)} = -1 \times (1.1)_2 \times 2^{65} = -(11)_2 \times 2^{64} = -3 \times 2^{64}$$

4.

S	Exponent	Significand
0	1000 0001	1010 0000 0000 0000 0000 000

$$(-1)^0 \times (1 + .101)_2 \times 2^{(129-127)} = (1.101)_2 \times 2^2 = (110.1)_2 = 6.5$$

HW7

Problem2

In a direct-mapped cache, each memory address is associated with one possible block within the cache. The memory address is divided into three fields, as shown below.

Tag	Index	(Byte) Offset

Suppose we have a 128Bytes of data in a direct-mapped cache with 16-byte blocks. Determine the size of the Tag, Index and (Byte) Offset fields if the memory address is 32-bit.

Answer:

The cache has 16-byte block , so the Offset is 4 bits (2^4 = 16).

The cache contains 128Bytes data (2^7 Bytes), and block contains 16 Bytes, so the direct-mapped cache has 8 (128/16) blocks. The Index needs 3 (2^3 = 8) bits.

Tag length = 32 – Index bits – Offset bits = 32 – 3 -4 = 25 bits

HW7

Problem3

For a direct-mapped cache design with a 32bit address, the following bits of the address are used to access the cache.

Tag	Index	Offset
31-13	12-6	5-0

a)What is the cache line size(in words)?

b)How many entries does the cache have?

c)What is the ratio between total bits required for such a cache implementation over the data storage bits? (Note the structure of the direct-mapped cache!)

Answer:

a) RISC-V 32位来说， 1 word = 32 bits = 4 bytes
cache line(block) size: 2^(Offset bits) = 2^6 = 64 bytes = 16 words

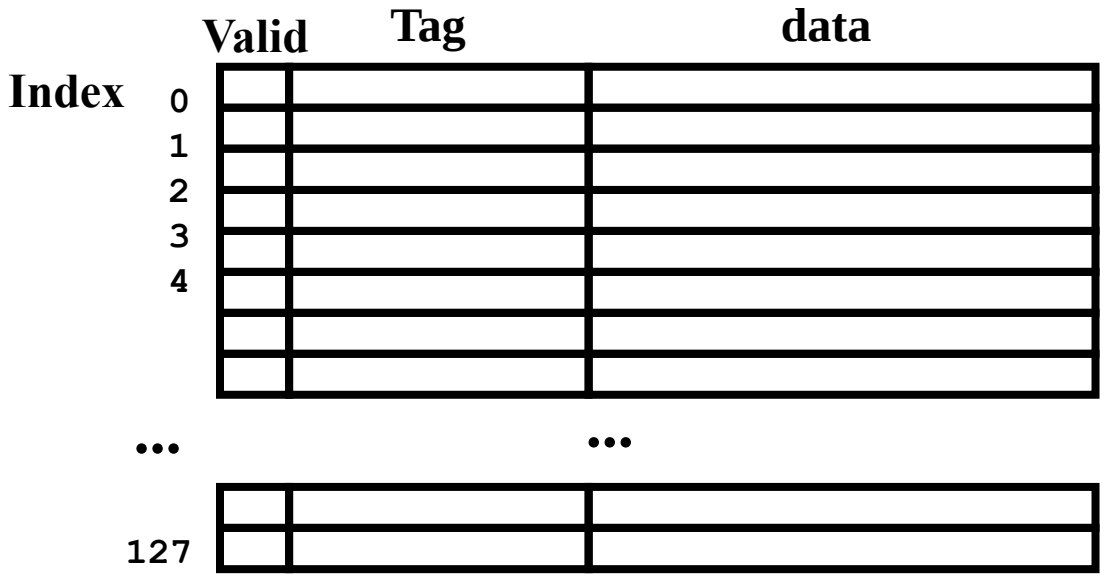
b) The number of entries: 2^(Index bits) = 2^7 = 128

HW7

Problem3

c)What is the ratio between total bits required for such a cache implementation over the data storage bits? (Note the structure of the direct-mapped cache!)

The structure of cache is shown as follows:



c) (Valid bits + Tag bits + data bits) / (data bits)
= (1 + 19 + 16*32) / 16*32 = 1.039

ISEE

- 谢谢大家！